TraceX®

User’s Manual: Software

Renesas Synergy™ Platform

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website.

Renesas Electronics
www.renesas.com

Oct 18, 2018
1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.

3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.

5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.

6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.

11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) "Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)
Renesas Synergy Specific Information

If you are using ThreadX for the Renesas Synergy platform, please use the following information.

Using TraceX with a Synergy Project

NOTE: Before you can use TraceX with your Synergy Project, you must download and install TraceX from the Synergy Gallery Website.

TraceX™ is a host-based analysis tool that provides a graphical view of real-time system events. TraceX collects data on the target device and displays the data for inspection and analysis. A TraceX version for Synergy devices is available for downloading on the Synergy Gallery Website and can be used together with the Synergy License.

To use TraceX, do the following:

1) In e² studio, add the ThreadX source code to your project by going to the Threads tab, clicking the New Stack button in the Stacks pane, and selecting X-Ware > ThreadX > ThreadX Source.

2) Enable TraceX in the Properties Window of the thread using the Threads tab. Keep the default name for the TraceX buffer as g_tx_trace_buffer.
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Checking</td>
<td>Enabled (default)</td>
</tr>
<tr>
<td>Max Priorities</td>
<td></td>
</tr>
<tr>
<td>Minimum Stack</td>
<td></td>
</tr>
<tr>
<td>Timer Thread Stack Size</td>
<td></td>
</tr>
<tr>
<td>Timer Thread Priority</td>
<td></td>
</tr>
<tr>
<td>Trace Time Mask</td>
<td></td>
</tr>
<tr>
<td>Timer Process in ISR</td>
<td>Enabled (default)</td>
</tr>
<tr>
<td>Reactivate Inline</td>
<td>Disabled (default)</td>
</tr>
<tr>
<td>Stack Filling</td>
<td>Enabled (default)</td>
</tr>
<tr>
<td>Stack Checking</td>
<td>Disabled (default)</td>
</tr>
<tr>
<td>Preemption Threshold</td>
<td>Disabled (default)</td>
</tr>
<tr>
<td>Redundant Clearing</td>
<td>Enabled (default)</td>
</tr>
<tr>
<td>No Timer</td>
<td>Disabled (default)</td>
</tr>
<tr>
<td>Notify Callbacks</td>
<td>Disabled (default)</td>
</tr>
<tr>
<td>Inline Thread Resume Suspend</td>
<td>Disabled (default)</td>
</tr>
<tr>
<td>Not Interruptable</td>
<td>Disabled (default)</td>
</tr>
<tr>
<td>Event Trace</td>
<td>Enabled</td>
</tr>
<tr>
<td>Trace Buffer Name</td>
<td>g_tx_trace_buffer</td>
</tr>
<tr>
<td>Trace Buffer Size</td>
<td>65536</td>
</tr>
<tr>
<td>Trace Buffer Number of Registries</td>
<td>30</td>
</tr>
</tbody>
</table>
3) Set the path to the TraceX application in **Window > Preferences > Renesas > TraceX**. The default path to TraceX is C:\Express_Logic\TraceX_5.2.0\TraceX.exe.

4) Build your project (**Project > Build All**).

5) Connect your Synergy target board.

6) Start a debug session (**Run > Debug**)

7) In **Run > TraceX**, select **Launch TraceX Debugging**.
8) In the TraceX Debugging window, set **Buffer Start Address** to \&g\_tx\_trace\_buffer.

In the same window, set **Buffer Size (bytes)** to the buffer size selected in the Properties Window in step 2. The default is 65536.

![TraceX Debugging window](image)

**Buffer Start Address and Size**

9) Click **OK**.

10) Run your code (**Run > Resume**) to collect TraceX data.

11) Suspend execution of your code (**Run > Suspend**)

12) Observe the collected data in TraceX.

![TraceX Collected Data](image)
14) To collect further TraceX data:

- Resume execution of your code
- Suspend execution of your code
- Click Run > TraceX and select Update TraceX Data.
ThreadX system analysis tool
V5 for Windows

User Guide

Express Logic, Inc.
858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259
http://www.expresslogic.com
©2002-2010 by Express Logic, Inc.

All rights reserved. This document and the associated TraceX software are the sole property of Express Logic, Inc. Each contains proprietary information of Express Logic, Inc. Reproduction or duplication by any means of any portion of this document without the prior written consent of Express Logic, Inc. is expressly forbidden.

Express Logic, Inc. reserves the right to make changes to the specifications described herein at any time and without notice in order to improve design or reliability of TraceX. The information in this document has been carefully checked for accuracy; however, Express Logic, Inc. makes no warranty pertaining to the correctness of this document.

Trademarks

TraceX, NetX, USBX, StackX, preemption-threshold, Piconet, and UDP Fast Path are trademarks of Express Logic, Inc., and ThreadX and FileX are registered trademarks of Express Logic, Inc.

All other product and company names are trademarks or registered trademarks of their respective holders.

Warranty Limitations

Express Logic, Inc. makes no warranty of any kind that the TraceX products will meet the USER’s requirements, or will operate in the manner specified by the USER, or that the operation of the TraceX products will operate uninterrupted or error free, or that any defects that may exist in the TraceX products will be corrected after the warranty period. Express Logic, Inc. makes no warranties of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, with respect to the TraceX products. No oral or written information or advice given by Express Logic, Inc., its dealers, distributors, agents, or employees shall create any other warranty or in any way increase the scope of this warranty, and licensee may not rely on any such information or advice.

Part Number: 000-1454
Revision 5.2
Contents

About This Guide 7
- Organization 7
- Guide Conventions 9
- Customer Support Center 9
  Latest Product Information 9
  What We Need From You 10
  Where to Send Comments About This Guide 10

1 Introduction to TraceX 11
- TraceX Requirements 12
- TraceX Constraints 12

2 Installation and Use of TraceX 13
- Product Distribution 14
- TraceX Installation Directory 14
- TraceX Installation 15
- Using TraceX 24
- TraceX Examples 25

3 Description of TraceX 27
- Display Overview 28
- Title Bar 29
- Tool Bar 30
- Display Mode Tabs 32
- Sequential View Mode 33
4 TraceX Performance Analysis  55

- Performance Analysis  56
- Multi-Core CPU Utilization  57
- Execution Profile  57
- Popular Services  58
- Thread Stack Usage  60
- Performance Statistics  61
- FileX Statistics  63
- NetX Statistics  65
- Trace File Information  66
- Raw Trace Dump  67

5 Generating Trace Buffers  69

- ThreadX Event Trace Support  70
- Enabling Event Trace  70
- Defining Time-Stamp Constants  70
Contents

- Exporting the Trace Buffer 72
- Extended Event Trace API 73

6 ThreadX Trace Events 95
- List of Events and Icons 96
- Event Descriptions 100

7 FileX Trace Events 117
- List of Events and Icons 118
- Event Descriptions 121

8 NetX Trace Events 135
- List of Events and Icons 136
- Event Descriptions 143

9 USBX Trace Events 171
- List of Events and Icons 172
- Event Descriptions 182

10 Customer User Events 221
- Inserting User-Defined Events 222
- Default Display of User-Defined Events 222
- Defining Custom User-Defined Event Icons 224

11 Format of Event Trace Buffer 231
- Event Trace Format 232
- Event Trace Control Header 232
  Control Header ID 233
  Timer Valid MASK 233

Express Logic, Inc.
Trace Base Address 234
Registry Start and End Pointers 234
Registry Name Size 234
Buffer Start and End Pointers 234
Current Buffer Pointer 235

Event Trace Object Registry 235
Object Available Flag 235
Object Entry Type 236
Object Pointer 237
Object Reserved Fields 237
Object Parameters 237
Object Name 237

Event Trace Entries 238
Thread Pointer 238
Thread Priority 239
Event ID 239
Information Fields (1-4) 239

A  Sample tx_port.h 241

B  tx_trace.h File 247

C  DOS Command Line Utilities 255

D  Dumping the Trace Buffer 257
  BenchX Tools 258
  RealView Tools 259
  IAR Tools 259
  CodeWarrior Tools 260
  MPLAB Tools 261
  GHS Tools 267
  Renesas HEW 268

Index 271
This guide contains comprehensive information about TraceX™, the Microsoft Windows-based system analysis tool from Express Logic, Inc.

It is intended for the embedded real-time software developer using ThreadX Real-Time Operating System (RTOS) and add-on components. The developer should be familiar with standard ThreadX FileX, and NetX concepts.

Organization

Chapter 1
Contains an basic overview of TraceX and describes its relationship to real-time development.

Chapter 2
Gives the basic steps to install and use TraceX to analyze your application right out of the box.

Chapter 3
Describes the main features of TraceX.

Chapter 4
Details performance analysis features of TraceX.

Chapter 5
Describes how to set up ThreadX, FileX, and NetX in order to generate a trace buffer that is viewable by TraceX.
Chapter 6  Describes TraceX events in detail.

Chapter 7  Describes FileX events in detail.

Chapter 8  Describes NetX events in detail.

Chapter 9  Describes USBX events in detail.

Chapter 10 Describes creating custom user events in detail.

Chapter 11 Describes the internal trace buffer in detail.

Appendix A  ThreadX port-specific file with its time-stamp source for gathering trace events.

Appendix B  ThreadX tx_trace.h file that shows implementation details regarding the event trace buffer.

Appendix C  Summarizes command line utilities for converting various file formats into proper TraceX binary files.

Appendix D  Examples of dumping trace files from various development tools.

Index  Topic cross reference.
Guide Conventions

*Italic* Typeface denotes book titles, emphasizes important words, and indicates variables.

**Boldface** Typeface denotes file names, key words, and further emphasizes important words and variables.

Indicates a specific area of interest in screenshots of the TraceX graphic user interface (GUI).

Indicate information of note.

Customer Support Center

<table>
<thead>
<tr>
<th>Support engineers</th>
<th>858.613.6640</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support fax</td>
<td>858.521.4259</td>
</tr>
<tr>
<td>Support email</td>
<td><a href="mailto:support@expresslogic.com">support@expresslogic.com</a></td>
</tr>
<tr>
<td>Web page</td>
<td><a href="http://www.expresslogic.com">http://www.expresslogic.com</a></td>
</tr>
</tbody>
</table>

Latest Product Information

Visit the Express Logic web site and select the “Support” menu option to find the latest support information, including information about the latest TraceX product releases.
What We Need From You

To more efficiently resolve your support request, provide us with the following in your email request:

- A detailed description of the problem, including frequency of occurrence and whether it can be reliably reproduced.
- An attached copy of the trace file causing the problem.
- The version of TraceX you are using.
- The version of ThreadX you are using as well as the `_tx_version_id` and `_tx_build_options` variables in ThreadX.
- The version of FileX you are using including the `_fx_version_id` string.
- The version of NetX you are using including the `_nx_version_id` string.

Where to Send Comments About This Guide

The staff at Express Logic is always striving to provide you with better products. To help us achieve this goal, email any comments and suggestions to the Customer Support Center at support@expresslogic.com

Enter “TraceX User Guide” in the subject line.
TraceX is a Microsoft system analysis tool that displays system event information gathered by ThreadX running on an embedded target. The user is responsible for transferring the trace buffer stored in RAM in the embedded target to a binary file on the host computer. The user can then open this file with TraceX and graphically analyze the target events, diagnosing system problems and tuning a working application to improve performance and resource management.

This chapter describes the following:

- TraceX Requirements 12
- TraceX Constraints 12
Introduction to TraceX

TraceX Requirements

TraceX requires Windows XP (or above). The system should have a minimum of 192MB of RAM, 2 GB of available hard-disk space, and a minimum display of 1024x768 with 256 colors. In addition, the application must be running on ThreadX V5.0 or later.

TraceX also requires the Microsoft .NET framework be installed, which the TraceX installer does automatically.

TraceX Constraints

TraceX has the following constraints:

- TraceX files are limited to a maximum of 32,768 events (roughly 1MB).
- The time-stamp source must have reasonable resolution. If the resolution is too low, the events will overlap. If the resolution is too high, there is potential for long gaps between events.
- TraceX cannot accurately measure intervals between events greater than the timer period.
This chapter contains a description of various issues related to installation, setup, and usage of the TraceX system analysis tool, including the following:

- Product Distribution 14
- TraceX Installation Directory 14
- TraceX Installation 15
- Using TraceX 24
- TraceX Examples 25
Product Distribution

TraceX is shipped on a single CD-ROM compatible disk. The package includes an installation program Setup.exe that automatically runs from the CD. If the TraceX installer does not automatically run, click on the Setup.exe program manually to install TraceX. The TraceX package also contains an example directory of pre-built traces that should serve as a good starting point for new TraceX users.

The release notes associated with each new TraceX release can be found in the file readme_tracex.txt. Review this file to see what has changed between successive TraceX releases.

TraceX Installation Directory

TraceX, by default, is installed in the directory c:\ExpressLogic\TraceX_v, where v is the version of TraceX being installed. The default location for TraceX installation may be changed via the installation dialog as shown in the next section.

TraceX requires the Microsoft .NET framework to operate. The installation of this is done automatically by the TraceX installer via the dialogs shown later in this chapter.
TraceX Installation

TraceX is easily installed, as shown in Figure 2.1 through Figure 2.7. The installation dialogs are fairly straightforward, but it is worth noting that Figure 2.4 shows the dialog for changing the default installation directory for TraceX.
Selecting **Next** button launches the TraceX installation, as shown in **Figure 2.2**.

![FIGURE 2.2](image)

Selecting **Next** button in **Figure 2.2** indicates the terms of the license agreement are agreed and
TraceX installation continues, as shown in the Figure 2.3.

![Figure 2.3](image)

At this point, the **User Name** and **Organization** should be entered, followed by selection of the **Next**
button, which continues the installation, as shown in the Figure 2.4.
If the default installation path is okay, simply select the **Next** button to continue the installation, as shown in **Figure 2.5**.

**FIGURE 2.5**
If everything is acceptable, select the **Next** button to continue the installation, as shown in **Figure 2.6**.
You should now observe the installation of TraceX on your Windows computer.

Selecting **Finish** button completes the installation and by default launches TraceX. At this point, TraceX is installed and ready to use!

As mentioned previously, TraceX requires .NET v3.5 or higher. If this is not currently installed, the TraceX installation process will automatically install it. **Figure**
2.8 through Figure 2.10 show the general flow of .NET installation.
Selecting **Install** button launches the Microsoft .NET installation, as shown in **Figure 2.9**.

![FIGURE 2.9](image)

Further note that the Microsoft .NET framework installation may take as much as eight minutes to complete. After complete, the Microsoft .NET installation requires a reboot. After the reboot, the installation will automatically continue where it left off. If it does not, launch the installation again.
Using TraceX

Using TraceX is as easy as opening a trace file inside TraceX! Run TraceX via the Start button. At this point you will observe the TraceX graphic user interface (GUI). You are now ready to use TraceX to graphically view an existing target trace buffer. This is easily done by clicking File -> Open, then entering the binary trace file.

You can also double-click on any trace file with an extension of trx, which will automatically launch TraceX.

Refer to Chapter 5 for instructions on how to generate trace buffers on the target using ThreadX.
TraceX Examples

A series of example trace files with the extension trx are found in the TraceFiles subdirectory of your installation. These pre-built examples will help you get comfortable with using TraceX on the trace buffers generated by ThreadX running with your application.

One example trace file always present is the file demo_threadx.trx. This example trace file shows the execution of the standard ThreadX demo, as described in Chapter 6 of the ThreadX User Guide.

FIGURE 2.11
Description of TraceX

This chapter describes the overall functionality of the TraceX system analysis tool, including the overall functionality of its GUI, including the following:

- Display Overview 28
- Title Bar 29
- Tool Bar 30
- Display Mode Tabs 32
- Sequential View Mode 33
- Time View Mode 34
- System Summary Line 35
- System Contexts 36
- Thread Status Information 40
- Event Information Display 42
- Current Event Display 44
- Event Searching 46
- Zooming In and Out 47
- Delta Ticks Between Events 49
- Actual Time Display 50
- Priority Inversions 51
- TraceX Multi-Core 53
Display Overview

Figure 3.1 shows the main display window of the TraceX system analysis tool. The layout is straightforward—the execution contexts are represented by the vertical elements on the left side; e.g., initialization, interrupt, idle, and the various thread entries. The events that take place in each context are displayed horizontally on the same context line. For example, the QR events shown below show that thread 2 is making successive calls to tx_queue_receive.

Context changes are represented by the vertical black lines that connect the context lines. The currently selected event is represented by a solid red vertical line. In this example, event 494 is selected.
Title Bar

The TraceX title bar provides several pieces of useful information. First is the current version of TraceX. Second is the full path of the currently opened trace file. The example in Figure 3.2 shows TraceX version 5.0.1 is displaying the demo_threadx.trx trace file.

FIGURE 3.2
Tool Bar

The TraceX tool bar provides several buttons to open trace files and control elements of their display. Figure 3.3 identifies the TraceX tool bar.

FIGURE 3.3

The TraceX tool bar buttons—from left to right—are defined as follows:
<table>
<thead>
<tr>
<th>Button</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Folder" /></td>
<td>Open a trace file</td>
</tr>
<tr>
<td><img src="image" alt="CPU" /></td>
<td>Generate multi-core CPU utilization analysis [Multi-Core Only]</td>
</tr>
<tr>
<td><img src="image" alt="Chart" /></td>
<td>Generate execution profile</td>
</tr>
<tr>
<td><img src="image" alt="Statistics" /></td>
<td>Generate performance statistics</td>
</tr>
<tr>
<td><img src="image" alt="Database" /></td>
<td>Generate Thread Stack usage</td>
</tr>
<tr>
<td><img src="image" alt="Magnifying Glass" /></td>
<td>Display currently selected event</td>
</tr>
<tr>
<td><img src="image" alt="Search" /></td>
<td>Search for events</td>
</tr>
<tr>
<td><img src="image" alt="Information" /></td>
<td>Open the TraceX User Guide.</td>
</tr>
<tr>
<td><img src="image" alt="Zoom In" /></td>
<td>Zoom in.</td>
</tr>
<tr>
<td><img src="image" alt="Zoom Out" /></td>
<td>Select percentage of display zoom, where 100% means the entire trace file is displayed within the current view.</td>
</tr>
<tr>
<td><img src="image" alt="Zoom Out" /></td>
<td>Zoom out.</td>
</tr>
</tbody>
</table>
Select first event.

Display previous event page.

Display previous event.

Determine how the next/previous navigation buttons operate. If **Event** is selected, navigation is done on the next/previous event. If **Context** is selected, navigation is done on the next/previous event on the specified context. If **Object** is selected, navigation is done on the next/previous event of the specified object; e.g., events associated with a specific queue. If **Switches** is selected, navigation is done on the next/previous change in context. If **ID** is selected, navigation is done on the next/previous event of the specified event ID.

Display next event.

Display next event page.

Select last event.

**Display Mode Tabs**

TraceX displays system events in two different ways: *sequential* and *time relative*. The default mode is sequential and that is the mode shown in **Figure 3.4**.
Changing the mode is as simple as selecting the *Sequential View* or *Time View* tabs in the TraceX window. Figure 3.4 shows the *Sequential View* and *Time View* tabs.

**Sequential View Mode**

The sequential view mode is selected by the *Sequential View* tab shown in Figure 3.4. This is the default mode. In this mode, events are shown immediately following each other, regardless of the elapsed time between them. Note also the ruler above the display area in Figure 3.4. It shows the relative event number from the beginning of the trace.
This mode is the default mode and is useful in getting a good overview of what is going on in the system.

**Time View Mode**

The time view mode is selected by the *Time View* button. **Figure 3.5** shows the same event trace as **Figure 3.4** except in time view mode. In this mode, events are shown in a time relative manner, with the solid green bar being used to show execution between events. This mode is useful to see where the bulk of processing is taking place in the system, which can help developers tune their system for greater performance and/or responsiveness.

**FIGURE 3.5**
Note also the ruler above the event display in Figure 3.5. This ruler shows relative ticks from the beginning of the trace, as derived from the time stamp instrumented in the event trace logging inside of ThreadX. If the time stamps are too close (low frequency timer), the events will run together. Conversely, if the time stamps are too far apart (high frequency timer), then the events will be too far apart. Choosing the right frequency time stamp is an important consideration in making the time relative view meaningful.

System Summary Line

TraceX also provides a single summary line (the top context in Figure 3.6) that includes all events on the same line. This makes it easy to see an overview of a complex system. The summary bar is especially beneficial in systems that have many threads. Without such a summary line, you would have to
follow complex system interactions using the vertical scroll bar to follow the context of execution.

![TraceX User Guide](image)

**FIGURE 3.6**

The summary line contains a summary of the context as well as the corresponding event summary underneath. In the example shown in **Figure 3.6**, it is easy to see that *thread 2* is executing and interrupted. The interrupt results in preemption by *thread 3*, *thread 6*, *thread 4*, and *thread 7*, after which *thread 2* resumes execution.

**System Contexts**

TraceX lists the system contexts on the left-hand side of the display, as shown in **Figure 3.7**. Events that occur in a particular context are displayed on the horizontal line to the right of that context. In this way, you can easily ascertain which context the event
occurred as well as follow that context line to see all the events that occurred in a particular context.

The first two context entries are always the Interrupt and Initialize/Idle contexts. Interrupt context represents all system events made from Interrupt Service Routines (ISRs). Initialize/Idle context represents two contexts in ThreadX. Events that occur during tx_application_define, are Initialize events and are displayed on the Initialize/Idle context. If the system is idle and thus no events are occurring, the green bar representing Running in the time view is drawn on the Initialize/Idle context.

![FIGURE 3.7](image)

In the example in Figure 3.7, there are nine thread contexts, starting from the System Timer Thread context. Additional information about an individual context is available by placing the mouse on that context. The additional information includes the thread’s starting stack address, ending stack.
address, total size, percent used, relative execution percentage, number of suspension, resumptions, and its highest and lowest priority during the trace. **Figure 3.8** shows information for **thread 0**.

**FIGURE 3.8**

Contexts may also be moved to group those of greater interest. This is accomplished by dragging and dropping the context or right-clicking on the context. Right-clicking on the context yields a dialog
for moving the context to the top or the bottom. 

Figure 3.9 shows the reorder dialog for thread 3.
Selecting Move to top results in the thread 3 context being moved to the top of the context list, as shown in Figure 3.10.

When enabled, TraceX displays the status of each thread via a colored line on the thread’s context. A green line indicates that the thread is in a “ready” state, while a line of any other color indicates the thread is suspended. For suspended threads, the color of the line indicates the type of ThreadX object that the thread is suspended on. For example, in Figure 3.11 the green line on the System Timer Thread’s context starting at event 496 shows that the System Timer Thread is ready. Prior to event 496 and after event 502, the absence of the green line
indicates that the **System Timer Thread** is suspended.

![FIGURE 3.11](image)

There are three modes of thread status display, available via the *Options -> Status Lines* menu as shown in **Figure 3.12**. The **Ready Only** option only shows the ready (green) status lines, but does not display any suspension status lines. This is the default option for TraceX. The **All On** option enables the display of all status lines (ready and suspension).
Finally, the **All Off** option disables the display of all status lines.

![Event Information Display](image)

**FIGURE 3.12**

**Event Information Display**

TraceX provides detailed information on some 600 run-time events, including ThreadX, FileX, NetX, NetX Duo, and USBX API calls and internal events. TraceX also supports up to an additional 61,439 unique user-defined events.

Regardless of whether sequential or time display mode is selected, a mouse-over on any event in the display area results in detailed event information displayed near the event. The mouse-over of event
494 in the demonstration `demo_threadx.trx` trace file is shown in Figure 3.13:

Each event displayed contains standard information about **Context** and both the **Relative Time** and **Time Stamp**. The Context field shows what context the event took place in. There are exactly four contexts: thread, idle, ISR, and initialization. When an event takes place in a thread context, the thread name and its priority at that time is gathered and displayed as shown above. The **Relative Time** shows the relative number of timer ticks from the beginning of the trace. The **Raw Time Stamp** displays the raw time source of the event. Finally, all event-specific information is displayed. This information is detailed throughout the remainder of this chapter.
Detailed event information is also available by double clicking on any event. Double clicking on event 494 is shown in Figure 3.14:

![Figure 3.14](image)

Being able to view multiple events at once gives the user a much richer view of what happened. Seeing them side by side is quite useful since many events are interrelated. This is accomplished by double-clicking on multiple events.

**Current Event Display**

TraceX displays the current event—in a separate window—when selected by the user via **View -> Current Event** or clicking on the current event button on the toolbar, as shown in Figure 3.15. After selected, TraceX displays the currently selected
event in a stand-alone window and refreshes this window whenever another event is selected. For example, Figure 3.16 shows the contents for currently selected event 494.

FIGURE 3.15

FIGURE 3.16
Event Searching

TraceX provides an extensive event search capability. The event ID and information fields of each event are the primary search parameters. Not specifying a value for a search parameter indicates that parameter effectively removes that parameter from the search. In addition, the search can be done such that any parameter found will satisfy the search or all parameters must be found to satisfy the search. The search may also be restricted to a particular context or cover all contexts in the trace. Invoking the event search is done by selecting the **Search by Value** button on the toolbar, as shown in [Figure 3.17](#). When selected the search dialog is displayed, which specifies all the parameters for the search. The **Next** and **Previous** buttons in the search dialog can then be used to find the next and previous events that match the specified search criteria. [Figure 3.18](#) shows the search dialog.

![FIGURE 3.17](image-url)
Zooming In and Out

By default, TraceX displays the events at their full size. You may zoom in or zoom out as desired. Zooming out is useful to see the overall events captured in the trace, while zooming in is useful in conditions where the events overlap because of the resolution of the time stamp source. Figure 3.19
shows the `demo_threadx.trx` file zoomed out so that 100% of the trace file is shown.

![ TRACEX User Guide](Image)

**FIGURE 3.19**

When zoomed out at 100% to show the entire trace within the current display page, it is easy to see all the context execution captured in the trace as well as the general events occurring within those contexts. Notice in Figure 3.12 that **thread 1** and **thread 2** execute most often. The blue coloring for their events also suggests that these threads are making queue service calls (queue events are blue in color).

Restoring to a full icon view is equally easy; Either the zoom-in button may be selected repeatedly or some factor of 100 may be entered.
Delta Ticks Between Events

Determining the number of ticks between various events in TraceX is easy—click on the starting event and drag the mouse to the ending event. The delta number of ticks between the events shows up in the upper right-hand corner of the display, as shown in Figure 3.20.

The delta ticks shown in Figure 3.20 show that 501 ticks have elapsed between event 494 and event 496. This could also be calculated manually by looking at the relative time stamps in each event and subtracting, but using the GUI is easy and instantaneous.
Actual Time Display

When enabled, TraceX displays the actual time in microseconds in **Time View** and for the various delta time information displayed by TraceX. By default, the actual time display is disabled. To enable the actual time display, the number of ticks per microsecond must be entered via the **Options -> Ticks per Microsecond** menu selection (the value to enter is determined by the hardware timer source used for the TraceX event logging on the target). Figure 3.21 shows a selection of 80 ticks per microsecond. After this selection, the TraceX tick information is also translated into microseconds, as shown in Figure 3.22.

![FIGURE 3.21](image-url)
Priority Inversions

TraceX automatically displays priority inversions detected in the trace file. Priority inversions are defined as conditions where a higher-priority thread is blocked trying to obtain a mutex that is currently owned by a lower-priority thread. This condition is termed **deterministic**, because the system was set up to operate in this manner. To inform the user, TraceX shows **deterministic** priority inversion ranges as a light salmon color.

TraceX also displays **non-deterministic** priority inversions. These priority inversions differ from the **deterministic** priority inversions in that another thread of a different priority level has executed in the middle of what was a **deterministic** priority inversion, thereby...
making the time within the priority inversion somewhat \textit{non-deterministic}. This condition is often unknown to the user and can be very serious. In order to alert the user of this condition, TraceX shows \textit{non-deterministic} priority inversions as a brighter salmon color. \textbf{Figure 3.23} shows both \textit{deterministic} and \textit{non-deterministic} priority inversions.

\textbf{Figure 3.15} shows a \textit{deterministic} priority inversion from event 24 through event 27. In this range, the higher-priority \texttt{thread 0} blocks on a mutex owned by a lower-priority \texttt{thread 1}. At event 27, \texttt{thread 1} releases the mutex and thus ends the priority inversion.

The brighter shaded area shows a \textit{non-deterministic} priority inversion between event 34 through event 45. What makes this \textit{non-deterministic} is that while \texttt{thread 1} holds the mutex that higher-priority \texttt{thread}
0 is blocked on, an interrupt occurs that resumes thread 2, which then executes and lengthens the time the system is in priority inversion. This condition can be quite serious and difficult to identify; however, with TraceX it is easily identified.

TraceX Multi-Core

TraceX Multi-Core is nearly identical to the standard TraceX, with two major additions. First, TraceX Multi-Core has the ability to display ThreadX SMP trace events on multiple cores. Second, TraceX Multi-Core introduces a new performance analysis capability. This new capability calculates the CPU Utilization of
Description of TraceX

Each core. Figure 3.24 shows a trace the standard ThreadX demonstration running on a ThreadX SMP platform.

![Figure 3.24](image)

Especially take note of the core identification on the left side. Each core contains the complete list of contexts such that all activities for all cores may be viewed together. In this example, event number 1501 shows queue receive event in Core 0, the next event 1502 shows a queue send event on Core 1.

Also note the new button for generating the CPU Utilization, as pointed to in Figure 3.24. Selecting this button produces a report defined in greater detail in Chapter 4.
This chapter describes the TraceX performance analysis tool:

- Performance Analysis 56
- Multi-Core CPU Utilization  57
- Execution Profile 57
- Popular Services 58
- Thread Stack Usage  60
- Performance Statistics  61
- FileX Statistics 63
- NetX Statistics 65
- Trace File Information 66
- Raw Trace Dump 67
Performance Analysis

TraceX provides built-in performance analysis of trace files. Information such as the execution profile, popular services, thread stack usage, and various performance statistics, including FileX and NetX statistics, are readily available. This information is available via the View menu item as shown in Figure 4.1.
Multi-Core CPU Utilization

Selecting the *CPU Utilization* button or *View -> CPU Utilization* presents the TraceX Multi-Core CPU utilization analysis for all cores in the currently loaded trace file. The CPU utilization associated with the sample ThreadX SMP demonstration running on two cores is shown in Figure 4.2.

![TraceX CPU Utilization](image)

**FIGURE 4.2**

Execution Profile

Selecting the *Generate Execution Profile* button or *View -> Execution Profile* presents the TraceX execution profile for the currently loaded trace file. The
execution profile associated with the sample ThreadX demonstration trace is shown in Figure 4.3.

![TraceX Execution Profile](image)

**FIGURE 4.3**

The example shown in Figure 4.3 indicates that nearly 49% of the processing time is inside of `thread 2` and nearly 47% of the processing time is inside of `thread 1`. This is logical since the bulk of the trace shows these threads sending and receiving messages. The remaining execution contexts have only a small amount of execution time in this example.

### Popular Services

Selecting `View -> Popular Services` presents the popular services in the currently loaded trace file. By
default, this information is displayed for the entire system. However, the popular services for specific threads are also available. The popular services in the sample ThreadX demonstration trace are shown in Figure 4.4.

The example shown in Figure 4.4 indicates that \texttt{tx\_queue\_send} and \texttt{tx\_queue\_receive} are the two most popular services in this trace. This is consistent with the behavior of the standard ThreadX demonstration from which this trace was captured.
As mentioned previously, specific threads can be selected for this analysis. **Figure 4.5** shows this analysis for **thread 3**.

**FIGURE 4.5**

**Thread Stack Usage**

Selecting the **Generate Thread Stack Usage** button or **View -> Thread Stack Usage** presents the stack usage for each thread in the trace file. This is accomplished by ThreadX including the current thread stack pointer in many of the trace entries in the file. A stack usage of 100% indicates the stack has overflowed and must be corrected in the
application. If there is no thread execution within this trace file, the stack usage for that thread is shown at 0%. The thread stack usage in the sample ThreadX demonstration trace is shown in Figure 4.6.

The example shown in Figure 4.5 indicates that most threads in this trace have between 9% and 12% stack usage.

Performance Statistics

Selecting the Generate Performance Statistics button or View -> Performance Statistics presents the performance statistics of the currently loaded trace file. By default, this information is displayed for the entire system. However, the performance statistics are also available for each specific thread.
The performance statistics of the sample ThreadX demonstration trace are shown in Figure 4.7.

![Figure 4.7](image)

The example shown in Figure 4.8 indicates that there were 39 context switches in this trace file, as well as three thread preemptions, 36 thread suspensions, 36 thread resumptions, and seven interrupts. There were no priority inversions found in this trace file. Notice there are two categories of priority inversions, namely, deterministic and non-deterministic. Deterministic priority inversions are priority inversion in which a thread is blocked on a mutex owned by a lower priority thread. An non-deterministic priority inversion is where a different lower priority thread runs during a deterministic priority inversion. The later can cause unforeseen
timing behavior in the application and should be studied carefully.

As mentioned previously, specific threads can be selected for this analysis. Figure 4.8 shows this analysis specifically for thread 3.

*FileX Statistics*

Selecting *View -> FileX Statistics* presents the FileX performance statistics of the currently loaded trace file. This information is displayed for the entire system, on all opened media objects. The
The performance statistics of the sample FileX demonstration trace are shown in Figure 4.9.

The example shown in Figure 4.9 indicates there were 39 media opens, 39 media closes, 39 media flushes, 39 directory reads, 39 directory writes, and 39 directory cache misses. There were also 39 file opens, 39 file closes, 1092 bytes read, 1092 bytes written, 195 logical sector reads, 78 logical sector writes, and 195 logical sector cache misses.
NetX Statistics

Selecting View -> NetX Statistics presents the NetX performance statistics of the currently loaded trace file. This information is displayed for the entire system. The performance statistics of the sample NetX demonstration trace are shown in Figure 4.10.

![NetX Statistics Table]

FIGURE 4.10
The example shown in Figure 4.10 indicates there were no ARP, Ping, or UDP events, but there were 70 IP packets sent, 3,188 IP bytes sent, 69 IP packets received, and 3,156 IP bytes received. There were also eight TCP client connections, nine TCP server connections, nine TCP packets sent, 252 TCP bytes sent, nine TCP packets received, and 252 TCP bytes received.

Trace File Information

Selecting View -> Trace File Information presents some basic information about the opened trace file. This information includes the byte order of the file, size of the time source, maximum number of bytes for each object name, and the base address of all trace file pointers. Figure 4.11 shows the trace file information for the standard demo_threadx.trx trace file.

FIGURE 4.11
Raw Trace Dump

Selecting View -> Raw Trace Dump presents a dialog to name the file containing the raw trace dump. After the file name and path are entered, TraceX builds the raw trace file in text format and launches notepad.exe to display it. Figure 4.12 shows the raw trace file dump for the standard demo_threadx.trx trace file.

FIGURE 4.12
This chapter contains a description about how to build a TraceX event buffer and also describes the underlying format of the buffer. This includes the following:

- ThreadX Event Trace Support 70
- Enabling Event Trace 70
- Defining Time-Stamp Constants 70
- Exporting the Trace Buffer 72
- Extended Event Trace API 73
  - Enable event tracing 74
  - Filter specified events 76
  - Unfilter specified events 80
  - Disable event tracing 84
  - Insert ISR enter event 86
  - Insert ISR exit event 88
  - Register trace buffer full application callback 90
- Insert user event 92
ThreadX Event Trace Support

ThreadX provides built-in event trace support for all ThreadX services, thread state changes, and user-defined events. The ThreadX event-trace capability is primarily designed as a post-mortem tool to analyze the last “n” activities in the application. From this information, the developer may spot problems and/or potential targets of optimization.

TraceX graphically displays the event trace buffer built by ThreadX. The following describes how to build the buffer and describes the underlying format of the buffer.

Enabling Event Trace

To enable event trace, define the time-stamp constants, build the ThreadX library with TX_ENABLE_EVENT_TRACE defined, and enable tracing by calling the tx_trace_enable function.

Defining Time-Stamp Constants

The time-stamp constants are designed to provide the developer control over the time-stamp used in the event trace entries. The two time-stamp constants and their default values are as follows:

```c
#ifndef TX_TRACE_TIME_SOURCE
#define TX_TRACE_TIME_SOURCE ++_tx_trace_simulated_time
#endif
#ifndef TX_TRACE_TIME_MASK
#define TX_TRACE_TIME_MASK 0xFFFFFFFFUL
#endif
```

The above constants are defined in tx_port.h and create a “fake” time-stamp that simply increments by one on each
event. The following is an example of an actual time-
stamp definition:

```c
#ifndef TX_TRACE_TIME_SOURCE
#define TX_TRACE_TIME_SOURCE *(ULONG *) 0x13000004)
#endif
#ifndef TX_TRACE_TIME_MASK
#define TX_TRACE_TIME_MASK 0xFFFFFFFFUL
#endif
```

The above constants specify a 32-bit timer that is
obtained by reading the address 0x13000004. Most
application specific time-stamps should be setup in a
similar fashion.
Exporting the Trace Buffer

TraceX needs the trace buffer in a binary, Intel HEX, or Motorola S-Record file format on the host. The easiest way to accomplish this is to stop the target and instruct your debugger to dump the memory area you supplied to `tx_trace_enable` function into a file on the host.

Warning: be careful not to stop the target within a trace gathering code itself. Doing so can cause invalid trace information. If the program is halted within ThreadX, it is best to step over any trace insert macro before dumping the trace buffer.

Appendix D shows how to dump the trace buffer from within a variety of development tools.
Extended Event Trace API

When ThreadX is built with `TX_ENABLE_EVENT_TRACE` defined, the following new event trace APIs are available to the application:

- `tx_trace_enable`
  
  *Enable event tracing*

- `tx_trace_event_filter`
  
  *Filter specified event(s)*

- `tx_trace_event_unfilter`
  
  *Unfilter specified event(s)*

- `tx_trace_disable`
  
  *Disable event tracing*

- `tx_trace_isr_enter_insert`
  
  *Insert ISR enter trace event*

- `tx_trace_isr_exit_insert`
  
  *Insert ISR exit trace event*

- `tx_trace_buffer_full_notify`
  
  *Register trace buffer full application callback*

- `tx_trace_user_event_insert`
  
  *Insert user event*
**tx_trace_enable**

Enable event tracing

**Prototype**

```c
UINT tx_trace_enable(VOID *trace_buffer_start,
                     ULONG trace_buffer_size, ULONG registry_entries);
```

**Description**

This service enables event tracing inside ThreadX. The trace buffer and the maximum number of ThreadX objects are supplied by the application.

The ThreadX library and application must be built with **TX_ENABLE_EVENT_TRACE** defined in order to use event tracing.

**Input Parameters**

- `trace_buffer_start`: Pointer to the start of the user-supplied trace buffer.
- `trace_buffer_size`: Total number of bytes in the memory for the trace buffer. The larger the trace buffer, the more entries it is able to store.
- `registry_entries`: Number of application ThreadX objects to keep in the trace registry. The registry is used to correlate object addresses with object names. This is highly useful for GUI trace analysis tools.

**Return Values**

- **TX_SUCCESS** (0x00): Successful event trace enable.
- **TX_SIZE_ERROR** (0x05): Specified trace buffer size is too small. It must be large enough for the trace header, the object registry, and at least one trace entry.
**TX_NOT_DONE**  (0x20) Event tracing was already enabled.

**TX_FEATURE_NOT_ENABLED** (0xFF) System was not compiled with trace enabled.

**Allowed From**
Initialization and threads

**Example**

```c
UCHAR my_trace_buffer[64000];

/* Enable event tracing using the global “my_trace_buffer” memory and supporting a maximum of 30 ThreadX objects in the registry. */
status = tx_trace_enable(&my_trace_buffer, 64000, 30);

/* If status is TX_SUCCESS the event tracing is enabled. */
```

**See Also**

- `tx_trace_event_filter`, `tx_trace_event_unfilter`, `tx_trace_disable`,
- `tx_trace_isr_enter_insert`, `tx_trace_isr_exit_insert`,
- `tx_trace_buffer_full_notify`, `tx_trace_user_event_insert`
tx_trace_event_filter

Filter specified events

Prototype

UINT tx_trace_event_filter(ULONG event_filter_bits);

Description

This service filters the specified event(s) from being inserted into the active trace buffer. Note that by default no events are filtered after tx_trace_enable is called.

The ThreadX library and application must be built with TX_ENABLE_EVENT_TRACE defined in order to use event tracing.

Input Parameters

- **event_filter_bits**
  - Bits that correspond to events to filter.
  - Multiple events may be filtered by simply or-ing together the appropriate constants. Valid
constants for this variable are defined as follows:

```
TX_TRACE_ALL_EVENTS          0x0000007FF
TX_TRACE_INTERNAL_EVENTS     0x00000001
TX_TRACE_BLOCK_POOL_EVENTS   0x00000002
TX_TRACE_BYTE_POOL_EVENTS    0x00000004
TX_TRACE_EVENT_FLAGS_EVENTS  0x00000008
TX_TRACE_INTERRUPT_CONTROL_EVENT 0x00000010
TX_TRACE_MUTEX_EVENTS        0x00000020
TX_TRACE_QUEUE_EVENTS        0x00000040
TX_TRACE_SEMAPHORE_EVENTS    0x00000080
TX_TRACE_THREAD_EVENTS       0x00000100
TX_TRACE_TIME_EVENTS         0x00000200
TX_TRACE_TIMER_EVENTS        0x00000400
FX_TRACE_ALL_EVENTS          0x00000000
FX_TRACE_INTERNAL_EVENTS     0x00000001
FX_TRACE_MEDIA_EVENTS        0x00000002
FX_TRACE_DIRECTORY_EVENTS    0x00000004
FX_TRACE_FILE_EVENTS         0x00000008
NX_TRACE_ALL_EVENTS          0x00000080
NX_TRACE_INTERNAL_EVENTS     0x00000100
NX_TRACE_ARP_EVENTS          0x00000200
NX_TRACE_ICMP_EVENTS         0x00000400
NX_TRACE_HICMP_EVENTS        0x00000800
NX_TRACE_IP_EVENTS           0x00001000
NX_TRACE_PACKET_EVENTS       0x00002000
NX_TRACE_TCP_EVENTS          0x00004000
NX_TRACE_UDP_EVENTS          0x00008000
UX_TRACE_ALL_EVENTS          0x00010000
UX_TRACE_ERRORS              0x00020000
UX_TRACE_HOST_STACK_EVENTS   0x00040000
UX_TRACE_DEVICE_STACK_EVENTS 0x00080000
UX_TRACE_HOST_CONTROLLER_EVENTS 0x00100000
UX_TRACE_DEVICE_CONTROLLER_EVENTS 0x00200000
UX_TRACE_HOST_CLASS_EVENTS   0x00400000
UX_TRACE_DEVICE_CLASS_EVENTS 0x00800000
```

Return Values

**TX_SUCCESS** (0x00) Successful event filter.

**TX_FEATURE_NOT_ENABLED** (0xFF) System was not compiled with trace enabled.

Allowed From

Initialization and threads
Example

/* Filter queue and byte pool events from trace buffer. */
status = tx_trace_event_filter(TX_TRACE_QUEUE_EVENTS | TX_TRACE_BYTE_POOL_EVENTS);

/* If status is TX_SUCCESS all queue and byte pool events are filtered. */

See Also

tx_trace_enable, tx_trace_event_unfilter, tx_trace_disable,
tx_trace_isr_enter_insert, tx_trace_isr_exit_insert, tx_trace_buffer_full_notify,
tx_trace_user_event_insert
tx_trace_event_unfilter

Unfilter specified events

Prototype

UINT  tx_trace_event_unfilter(ULONG event_unfilter_bits);

Description

This service unfilters the specified event(s) such that they will be inserted into the active trace buffer.

The ThreadX library and application must be built with TX_ENABLE_EVENT_TRACE defined in order to use event tracing.

Input Parameters

- **event_unfilter_bits**
  - Bits that correspond to events to unfilter. Multiple events may be unfiltered by simply or-ing together the appropriate constants.
Valid constants for this variable are defined as follows:

```
<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX_TRACE_ALL_EVENTS</td>
<td>0x00000007FF</td>
</tr>
<tr>
<td>TX_TRACE_INTERNAL_EVENTS</td>
<td>0x00000001</td>
</tr>
<tr>
<td>TX_TRACE_BLOCK_POOL_EVENTS</td>
<td>0x00000002</td>
</tr>
<tr>
<td>TX_TRACE_BYTE_POOL_EVENTS</td>
<td>0x00000004</td>
</tr>
<tr>
<td>TX_TRACE_EVENT_FLAGS_EVENTS</td>
<td>0x00000008</td>
</tr>
<tr>
<td>TX_TRACE_INTERRUPT_CONTROL_EVENT</td>
<td>0x00000010</td>
</tr>
<tr>
<td>TX_TRACE_MUTEX_EVENTS</td>
<td>0x00000020</td>
</tr>
<tr>
<td>TX_TRACE_QUEUE_EVENTS</td>
<td>0x00000040</td>
</tr>
<tr>
<td>TX_TRACE_SEMAPHORE_EVENTS</td>
<td>0x00000080</td>
</tr>
<tr>
<td>TX_TRACE_THREAD_EVENTS</td>
<td>0x00000100</td>
</tr>
<tr>
<td>TX_TRACE_TIME_EVENTS</td>
<td>0x00000200</td>
</tr>
<tr>
<td>TX_TRACE_TIMER_EVENTS</td>
<td>0x00000400</td>
</tr>
<tr>
<td>TX_TRACE_ALL_EVENTS</td>
<td>0x000007FF</td>
</tr>
<tr>
<td>TX_TRACE_INTERNAL_EVENTS</td>
<td>0x00000800</td>
</tr>
<tr>
<td>TX_TRACE_BLOCK_POOL_EVENTS</td>
<td>0x00000800</td>
</tr>
<tr>
<td>TX_TRACE_BYTE_POOL_EVENTS</td>
<td>0x00001000</td>
</tr>
<tr>
<td>FX_TRACE_DIRECTORY_EVENTS</td>
<td>0x00002000</td>
</tr>
<tr>
<td>FX_TRACE_FILE_EVENTS</td>
<td>0x00004000</td>
</tr>
<tr>
<td>NX_TRACE_ALL_EVENTS</td>
<td>0x00008000</td>
</tr>
<tr>
<td>NX_TRACE_INTERNAL_EVENTS</td>
<td>0x00010000</td>
</tr>
<tr>
<td>NX_TRACE_ARP_EVENTS</td>
<td>0x00020000</td>
</tr>
<tr>
<td>NX_TRACE_ICMP_EVENTS</td>
<td>0x00040000</td>
</tr>
<tr>
<td>NX_TRACE_UDP_EVENTS</td>
<td>0x00080000</td>
</tr>
<tr>
<td>UX_TRACE_ALL_EVENTS</td>
<td>0x00100000</td>
</tr>
<tr>
<td>UX_TRACE_ERRORS</td>
<td>0x00200000</td>
</tr>
<tr>
<td>UX_TRACE_HARDWARE_EVENTS</td>
<td>0x00400000</td>
</tr>
<tr>
<td>UX_TRACE_TRACING_EVENTS</td>
<td>0x00800000</td>
</tr>
<tr>
<td>UX_TRACE_TRACING_EVENTS</td>
<td>0x01000000</td>
</tr>
<tr>
<td>UX_TRACE_TRACING_EVENTS</td>
<td>0x02000000</td>
</tr>
<tr>
<td>UX_TRACE_TRACING_EVENTS</td>
<td>0x04000000</td>
</tr>
<tr>
<td>UX_TRACE_TRACING_EVENTS</td>
<td>0x08000000</td>
</tr>
<tr>
<td>UX_TRACE_TRACING_EVENTS</td>
<td>0x10000000</td>
</tr>
<tr>
<td>UX_TRACE_TRACING_EVENTS</td>
<td>0x20000000</td>
</tr>
</tbody>
</table>
```

**Return Values**

- **TX_SUCCESS** (0x00) Successful event unfilter.
- **TX_FEATURE_NOT_ENABLED** (0xFF) System was not compiled with trace enabled.

**Allowed From**

Initialization and threads
Example

/* Un-filter queue and byte pool events from trace buffer. */
status =
    tx_trace_event_unfilter(TX_TRACE_QUEUE_EVENTS | TX_TRACE_BYTE_POOL_EVENTS);

/* If status is TX_SUCCESS all queue and byte pool events are un-filtered. */

See Also

    tx_trace_enable, tx_trace_event_filter, tx_trace_disable,
    tx_trace_isr_enter_insert, tx_trace_isr_exit_insert,
    tx_trace_buffer_full_notify, tx_trace_user_event_insert
**tx_trace_disable**

Disable event tracing

**Prototype**

```c
UINT tx_trace_disable(VOID);
```

**Description**

This service disables event tracing inside ThreadX. This can be useful if the application wants to freeze the current event trace buffer and possibly transport it externally during run-time. Once disabled, the `tx_trace_enable` can be called to start tracing again.

The ThreadX library and application must be built with `TX_ENABLE_EVENT_TRACE` defined in order to use event tracing.

**Input Parameters**

None.

**Return Values**

- **TX_SUCCESS** (0x00) Successful event trace disable.
- **TX_NOT_DONE** (0x20) Event tracing was not enabled.
- **TX_FEATURE_NOT_ENABLED** (0xFF) System was not compiled with trace enabled.

**Allowed From**

Initialization and threads
Example

/* Disable event tracing. */
status = tx_trace_disable();

/* If status is TX_SUCCESS the event tracing is disabled. */

See Also

tx_trace_enable, tx_trace_event_filter, tx_trace_event_unfilter,
tx_trace_isr_enter_insert, tx_trace_isr_exit_insert,
tx_trace_buffer_full_notify, tx_trace_user_event_insert
tx_trace_isr_enter_insert

Insert ISR enter event

Prototype

VOID tx_trace_isr_enter_insert(ULONG isr_id);

Description

This service inserts the ISR enter event into the event trace buffer. It should be called by the application at the beginning of ISR processing. The supplied parameter should identify the specific ISR to the application.

The ThreadX library and application must be built with TX_ENABLE_EVENT_TRACE defined in order to use event tracing.

Input Parameters

isr_id Application specific value to identify the ISR.

Return Values

None

Allowed From

ISRs
Example

/* Insert trace event to identify the application’s ISR with an 
   ID of 3. */
status = tx_trace_isr_enter_insert(3);

/* If status is TX_SUCCESS the ISR entry event was inserted. */

See Also

tx_trace_enable, tx_trace_event_filter, tx_trace_event_unfilter, 
tx_trace_disable, tx_trace_isr_exit_insert, tx_trace_buffer_full_notify, 
tx_trace_user_event_insert
tx_trace_isr_exit_insert

Insert ISR exit event

Prototype

VOID tx_trace_isr_exit_insert(ULONG isr_id);

Description

This service inserts the ISR entry event into the event trace buffer. It should be called by the application at the beginning of ISR processing. The supplied parameter should identify the ISR to the application.

The ThreadX library and application must be built with TX_ENABLE_EVENT_TRACE defined in order to use event tracing.

Input Parameters

isr_id

Application specific value to identify the ISR.

Return Values

None

Allowed From

ISRs
Example

/* Insert trace event to identify the application’s ISR with an ID of 3. */
status = tx_trace_isr_exit_insert(3);

/* If status is TX_SUCCESS the ISR exit event was inserted. */

See Also

tx_trace_enable, tx_trace_event_filter, tx_trace_event_unfilter,
tx_trace_disable, tx_trace_isr_enter_insert, tx_trace_buffer_full_notify,
tx_trace_user_event_insert
tx_trace_buffer_full_notify

Register trace buffer full application callback

Prototype

VOID tx_trace_buffer_full_notify(VOID (*full_buffer_callback)(VOID *));

Description

This service registers an application callback function that is called by ThreadX when the trace buffer becomes full. The application can then choose to disable tracing and/or possibly setup a new trace buffer.

The ThreadX library and application must be built with TX_ENABLE_EVENT_TRACE defined in order to use event tracing.

Input Parameters

full_buffer_callback Application function to call when the trace buffer is full. A value of NULL disables the notification callback.

Return Values

None

Allowed From

ISRs
Example

```c
/* Application specific processing goes here! */

/* Register the "my_trace_is_full" function to be called whenever the 
trace buffer fills. */
status = tx_trace_buffer_full_notify(my_trace_is_full);

/* If status is TX_SUCCESS the "my_trace_is_full" function is registered. */
```

See Also

- tx_trace_enable, tx_trace_event_filter, tx_trace_event_unfilter,
- tx_trace_disable, tx_trace_isr_enter_insert, tx_trace_isr_exit_insert,
- tx_trace_user_event_insert
tx_trace_user_event_insert

Insert user event

Prototype

UINT tx_trace_user_event_insert(ULONG event_id,
                                  ULONG info_field_1, ULONG info_field_2,
                                  ULONG info_field_3, ULONG info_field_4);

Description

This service inserts the user event into the trace buffer. User event IDs must be greater than the constant TX_TRACE_USER_EVENT_START, which is defined to be 4096. The maximum user event is defined by the constant TX_TRACE_USER_EVENT_END, which is defined to be 65535. All events within this range are available to the application. The information fields are application specific.

The ThreadX library and application must be built with TX_ENABLE_EVENT_TRACE defined in order to use event tracing.

Input Parameters

event_id Application-specific event identification and must start be greater than TX_TRACE_USER_EVENT_START and less than or equal to TX_TRACE_USER_EVENT_END.

info_field_1 Application-specific information field.

info_field_2 Application-specific information field.

info_field_3 Application-specific information field.

info_field_4 Application-specific information field.

Return Values

TX_SUCCESS (0x00) Successful user event insert.

TX_NOT_DONE (0x20) Event tracing is not enabled.

TX_FEATURE_NOT_ENABLED (0xFF) The system was not compiled with trace enabled.
Allowed From
Initialization and threads

Example

/* Insert user event 3000, with info fields of 1, 2, 3, 4. */
status = \texttt{tx\_trace\_user\_event\_insert}(3000, 1, 2, 3, 4);

/* If status is TX\_SUCCESS the user event was inserted. */

See Also

\texttt{tx\_trace\_enable}, \texttt{tx\_trace\_event\_filter}, \texttt{tx\_trace\_event\_unfilter},
\texttt{tx\_trace\_disable}, \texttt{tx\_trace\_isr\_enter\_insert}, \texttt{tx\_trace\_isr\_exit\_insert},
\texttt{tx\_trace\_buffer\_full\_notify}
This chapter describes the ThreadX events.

- List of Events and Icons 96
- Event Descriptions 100
List of Events and Icons

The following is a list of ThreadX events displayed by TraceX:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Internal thread resume" /></td>
<td>Internal thread resume</td>
</tr>
<tr>
<td><img src="image" alt="Internal thread suspend" /></td>
<td>Internal thread suspend</td>
</tr>
<tr>
<td><img src="image" alt="Interrupt Service Routine (ISR) Enter" /></td>
<td>Interrupt Service Routine (ISR) Enter</td>
</tr>
<tr>
<td><img src="image" alt="Interrupt Service Routine (ISR) Exit" /></td>
<td>Interrupt Service Routine (ISR) Exit</td>
</tr>
<tr>
<td><img src="image" alt="Internal time-slice" /></td>
<td>Internal time-slice</td>
</tr>
<tr>
<td><img src="image" alt="Running" /></td>
<td>Running</td>
</tr>
<tr>
<td><img src="image" alt="User-Defined Event" /></td>
<td>User-Defined Event (See Chapter 4)</td>
</tr>
<tr>
<td><img src="image" alt="Block pool allocate" /></td>
<td>Block pool allocate (<em>tx_block_allocate</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Block pool create" /></td>
<td>Block pool create (<em>tx_block_pool_create</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Block pool delete" /></td>
<td>Block pool delete (<em>tx_block_pool_delete</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Block pool information get" /></td>
<td>Block pool information get (<em>tx_block_pool_info_get</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Block pool performance information get" /></td>
<td>Block pool performance information get (<em>tx_block_pool_performance_info_get</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Block pool system performance information get" /></td>
<td>Block pool system performance information get (<em>tx_block_pool_performance_system_info_get</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Block pool prioritize" /></td>
<td>Block pool prioritize (<em>tx_block_pool_prioritize</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Block release to pool" /></td>
<td>Block release to pool (<em>tx_block_release</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Byte pool allocate memory" /></td>
<td>Byte pool allocate memory (<em>tx_byte_allocate</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Byte pool create" /></td>
<td>Byte pool create (<em>tx_byte_pool_create</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Byte pool delete" /></td>
<td>Byte pool delete (<em>tx_byte_pool_delete</em>)</td>
</tr>
<tr>
<td><img src="image" alt="Byte pool information get" /></td>
<td>Byte pool information get (<em>tx_byte_pool_info_get</em>)</td>
</tr>
</tbody>
</table>
Byte pool performance information get
(tx_byte_pool_performance_info_get)

Byte pool system performance information get
(tx_byte_pool_performance_system_info_get)

Byte pool prioritize (tx_byte_pool_prioritize)

Byte memory release to pool (tx_byte_release)

Event flags create (tx_event_flags_create)

Event flags delete (tx_event_flags_delete)

Event flags get (tx_event_flags_get)

Event flags information get (tx_event_flags_info_get)

Event flags performance information get
(tx_event_flags_performance_info_get)

Event flags system performance information get
(tx_event_flags_performance_system_info_get)

Event flags set (tx_event_flags_set)

Event flags set notify (tx_event_flags_set_notify)

Interrupt enable/disable (tx_interrupt_control)

Mutex create (tx_mutex_create)

Mutex delete (tx_mutex_delete)

Mutex get (tx_mutex_get)

Mutex information get (tx_mutex_info_get)

Mutex performance information get
(tx_mutex_performance_info_get)

Mutex system performance information get
(tx_mutex_performance_system_info_get)

Mutex prioritize (tx_mutex_prioritize)
Mutex put \((tx\_mutex\_put)\)

Queue create \((tx\_queue\_create)\)

Queue delete \((tx\_queue\_delete)\)

Queue flush \((tx\_queue\_flush)\)

Queue front send \((tx\_queue\_front\_send)\)

Queue information get \((tx\_queue\_info\_get)\)

Queue performance information get \((tx\_queue\_performance\_info\_get)\)

Queue system performance information get \((tx\_queue\_performance\_system\_info\_get)\)

Queue prioritize \((tx\_queue\_prioritize)\)

Queue receive message \((tx\_queue\_receive)\)

Queue send message \((tx\_queue\_send)\)

Queue send notify \((tx\_queue\_send\_notify)\)

Semaphore ceiling put \((tx\_semaphore\_ceiling\_put)\)

Semaphore create \((tx\_semaphore\_create)\)

Semaphore delete \((tx\_semaphore\_delete)\)

Semaphore get \((tx\_semaphore\_get)\)

Semaphore information get \((tx\_semaphore\_info\_get)\)

Semaphore performance information get \((tx\_semaphore\_performance\_info\_get)\)

Semaphore system performance information get \((tx\_semaphore\_performance\_system\_info\_get)\)

Semaphore prioritize \((tx\_semaphore\_prioritize)\)

Semaphore put \((tx\_semaphore\_put)\)
Semaphore put notify (tx_semaphore_put_notify)

Thread create (tx_thread_create)

Thread delete (tx_thread_delete)

Thread exit/entry notify
(tx_thread_entry_exit_notify)

Thread identify (tx_thread_identify)

Thread information get (tx_thread_info_get)

Thread performance information get
(tx_thread_performance_info_get)

Thread performance system information get
(tx_thread_performance_system_info_get)

Thread preemption change
(tx_thread_preemption_change)

Thread priority change (tx_thread_priority_change)

Thread relinquish (tx_thread_relinquish)

Thread reset (tx_thread_reset)

Thread resume (tx_thread_resume)

Thread Sleep (tx_thread_sleep)

Thread stack error notify
(tx_thread_stack_error_notify)

Thread suspend (tx_thread_suspend)

Thread terminate (tx_thread_terminate)

Thread time-slice change
(tx_thread_time_slice_change)

Thread wait abort (tx_thread_wait_abort)

Time get (tx_time_get)

Time set (tx_time_set)
Event Descriptions

The following describes each individual event.
### Internal thread resume

**Description**
This event represents the internal processing in ThreadX that resumes a thread for execution. If the specified thread is the highest priority and preemption-threshold does not block its execution, the system will start executing this newly ready thread.

**Information Fields**
- **Info Field 1**: Pointer to the thread being resumed.
- **Info Field 2**: Previous state of the thread being resumed, as follows:
  - TX_READY
  - TX_COMPLETED
  - TX_TERMINATED
  - TX_SUSPENDED
  - TX_SLEEP
  - TX_QUEUE_SUSP
  - TX_SEMAPHORE_SUSP
  - TX_EVENT_FLAG
  - TX_BLOCK_MEMORY
  - TX_BYTE_MEMORY
  - TX_TCP_IP
  - TX_MUTEX_SUSP
- **Info Field 3**: Stack pointer value during the call.
- **Info Field 4**: Pointer to next highest priority thread to execute.

### Internal thread suspend

**Description**
This event represents the internal processing in ThreadX that suspends a thread’s execution. The next highest priority thread ready for execution is placed in the fourth information field. If this value is NULL, there is no other thread ready for execution and the system is idle.

**Information Fields**
- **Info Field 1**: Pointer to the thread being suspended.
- **Info Field 2**: New state of the thread being suspended, as follows:
  - TX_COMPLETED
  - TX_TERMINATED
  - TX_SUSPENDED
  - TX_SLEEP
  - TX_QUEUE_SUSP
  - TX_SEMAPHORE_SUSP
  - TX_EVENT_FLAG
  - TX_BLOCK_MEMORY
  - TX_BYTE_MEMORY
  - TX_TCP_IP
  - TX_MUTEX_SUSP
- **Info Field 3**: Stack pointer value during the call.
- **Info Field 4**: Pointer to next highest priority thread to execute. If NULL, the system is idle.

### Interrupt Service Routine (ISR) enter

**Description**
This event represents entering an Interrupt Service Routine (ISR) in the application. The interrupt service routine execution continues until the ISR exit event takes place.

**Information Fields**
- **Info Field 1**: Stack pointer value during the call.
- **Info Field 2**: Application-defined ISR number (optional).
- **Info Field 3**: Nested interrupt count.
- **Info Field 4**: Internal preemption disable flag.

### Interrupt Service Routine (ISR) exit

**Description**
This event represents exiting an Interrupt Service Routine (ISR) in the application.

**Information Fields**
- **Info Field 1**: Stack pointer value during the call.
- **Info Field 2**: Application-defined ISR number (optional).
- **Info Field 3**: Nested interrupt count.
- **Info Field 4**: Internal preemption disable flag.
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Internal time-slice        | 🍒          | This event represents the internal processing in ThreadX that performs the time-slice operation. The next thread of the same priority is placed in the first information field. If this value is the same as the current thread, no time-slice was performed. | Info Field 1: Pointer to the next thread to execute.  
Info Field 2: Nested interrupt count.  
Info Field 3: Internal preemption disable flag.  
Info Field 4: Stack pointer value during the call. |
| Block Allocate             | 🔄          | This event represents allocating a memory block via tx_block_allocate. If successful, the address of the block allocated is returned in the second information field.                                           | Info Field 1: Pointer to the corresponding block pool.  
Info Field 2: Pointer to the memory block returned (if successful).  
Info Field 3: The wait option supplied to the tx_block_allocate call.  
Info Field 4: Remaining available blocks in the pool after this allocation. |
| Block Pool Create          | 🍍          | This event represents creating a memory block pool via tx_block_pool_create.                                                                                                                                   | Info Field 1: Pointer to the block pool control block.  
Info Field 2: Pointer to the starting memory area of the pool.  
Info Field 3: The number of blocks in the pool.  
Info Field 4: The size of each block in the pool in bytes. |
| Block Pool Delete          | 🍎          | This event represents deleting a memory block pool via tx_block_pool_delete.                                                                                                                                    | Info Field 1: Pointer to the block pool control block.  
Info Field 2: Stack pointer value during the call.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Block Pool Information Get | 🍏          | This event represents getting information about a memory block pool via tx_block_pool_info_get.                                                                                                             | Info Field 1: Pointer to the block pool control block.  
Info Field 2: Stack pointer value during the call.  
Info Field 3: Not used.  
Info Field 4: Not used. |
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block Pool Performance Information Get</strong></td>
<td><img src="icon1.png" alt="Icon" /></td>
<td>This event represents getting performance information about a memory block pool via tx_block_pool_performance_info_get.</td>
<td>Info Field 1: Pointer to the block pool control block.&lt;br&gt;Info Field 2: Not used.&lt;br&gt;Info Field 3: Not used.&lt;br&gt;Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Block Pool Performance System Information Get</strong></td>
<td><img src="icon2.png" alt="Icon" /></td>
<td>This event represents getting performance information about all memory block pools via tx_block_pool_performance_system_info_get.</td>
<td>Info Field 1: Not used.&lt;br&gt;Info Field 2: Not used.&lt;br&gt;Info Field 3: Not used.&lt;br&gt;Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Block Pool Prioritize</strong></td>
<td><img src="icon3.png" alt="Icon" /></td>
<td>This event represents placing the highest-priority suspended thread at the front of the block pool suspension list. If this is done prior to calling tx_block_release, the highest priority suspended thread will receive the released block.</td>
<td>Info Field 1: Memory block pool pointer.&lt;br&gt;Info Field 2: Number of threads suspended on this block pool.&lt;br&gt;Info Field 3: Stack pointer at the time of the call.&lt;br&gt;Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Block Release</strong></td>
<td><img src="icon4.png" alt="Icon" /></td>
<td>This event represents releasing a previously allocated block back to the block pool.</td>
<td>Info Field 1: Memory block pool pointer.&lt;br&gt;Info Field 2: Pointer to block to release.&lt;br&gt;Info Field 3: Number of threads suspended on this block pool.&lt;br&gt;Info Field 4: Stack pointer at the time of the call.</td>
</tr>
<tr>
<td><strong>Byte Allocate</strong></td>
<td><img src="icon5.png" alt="Icon" /></td>
<td>This event represents allocating memory via tx_byte_allocate. If successful, the address of the memory allocated is returned in the second information field.</td>
<td>Info Field 1: Pointer to the corresponding byte pool.&lt;br&gt;Info Field 2: Pointer to the memory returned (if successful).&lt;br&gt;Info Field 3: Number of bytes requested.&lt;br&gt;Info Field 4: The wait option supplied to the tx_byte_allocate call.</td>
</tr>
<tr>
<td><strong>Byte Pool Create</strong></td>
<td><img src="icon6.png" alt="Icon" /></td>
<td>This event represents creating a byte pool via tx_byte_pool_create.</td>
<td>Info Field 1: Pointer to the corresponding byte pool.&lt;br&gt;Info Field 2: Pointer to the start of the memory area.&lt;br&gt;Info Field 3: Number of bytes in the byte pool.&lt;br&gt;Info Field 4: The stack pointer at the time of the call.</td>
</tr>
</tbody>
</table>
Byte Pool Delete

**tx_byte_pool_delete**

**Description**
This event represents deleting a byte pool via `tx_byte_pool_delete`.

**Information Fields**
- Info Field 1: Pointer to the corresponding byte pool.
- Info Field 2: The stack pointer at the time of the call.
- Info Field 3: Not used.
- Info Field 4: Not used.

---

Byte Pool Information Get

**tx_byte_pool_info_get**

**Description**
This event represents getting byte pool information via `tx_byte_pool_info_get`.

**Information Fields**
- Info Field 1: Pointer to the corresponding byte pool.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

---

Byte Pool Performance Info Get

**tx_byte_pool_performance_info_get**

**Description**
This event represents getting byte pool performance information via `tx_byte_pool_performance_info_get`.

**Information Fields**
- Info Field 1: Pointer to the corresponding byte pool.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

---

Byte Pool Performance System Info Get

**tx_byte_pool_performance_system_info_get**

**Description**
This event represents getting byte pool performance system information via `tx_byte_pool_performance_system_info_get`.

**Information Fields**
- Info Field 1: Not used.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

---

Byte Pool Prioritize

**tx_byte_pool_prioritize**

**Description**
This event represents prioritizing the byte pool’s suspension list via `tx_byte_pool_prioritize`.

**Information Fields**
- Info Field 1: Pointer to corresponding byte pool.
- Info Field 2: Number of threads currently suspended on byte pool.
- Info Field 3: Stack pointer at time of call.
- Info Field 4: Not used.

---

Byte Release

**tx_byte_release**

**Description**
This event represents releasing a block of memory allocated from a byte pool via `tx_byte_release`.

**Information Fields**
- Info Field 1: Pointer to corresponding byte pool.
- Info Field 2: Pointer to previously allocated byte pool memory.
- Info Field 3: Number of threads suspended on this byte pool.
- Info Field 4: Number of available bytes of memory.
<table>
<thead>
<tr>
<th>Event Flags Create</th>
<th>Event Flags Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tx_event_flags_create</strong></td>
<td><strong>tx_event_flags_delete</strong></td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents creating a new event flags group via <code>tx_event_flags_create</code>.</td>
<td>This event represents deleting an event flags group via <code>tx_event_flags_delete</code>.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Pointer to event flags group control block.</td>
<td>Info Field 1: Pointer to event flags group.</td>
</tr>
<tr>
<td>Info Field 2: Stack pointer at time of call.</td>
<td>Info Field 2: Stack pointer at time of call.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event Flags Get</th>
<th>Event Flags Information Get</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tx_event_flags_get</strong></td>
<td><strong>tx_event_flags_info_get</strong></td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents retrieving event flags from an existing event flags group via <code>tx_event_flags_get</code>.</td>
<td>This event represents retrieving information regarding an existing event flags group via <code>tx_event_flags_info_get</code>.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Pointer to event flags group.</td>
<td>Info Field 1: Pointer to event flags group.</td>
</tr>
<tr>
<td>Info Field 2: Event flags requested.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Event flags currently set in the group.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Option requested on the event flags get.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event Flags Performance Information Get</th>
<th>Event Flags Performance System Info Get</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tx_event_flags_performance_info_get</strong></td>
<td><strong>tx_event_flags_performance_system_info_get</strong></td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents retrieving performance information regarding an existing event flags group via <code>tx_event_flags_performance_info_get</code>.</td>
<td>This event represents retrieving performance information regarding an existing event flags group via <code>tx_event_flags_performance_info_get</code>.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Pointer to event flags group.</td>
<td>Info Field 1: Not used.</td>
</tr>
<tr>
<td>Info Field 2: Not used</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>
**Event Flags Set**

- **Icon**: tx_event_flags_set
- **Description**: This event represents setting (or clearing) event flags in an existing event flags group via `tx_event_flags_set`.
- **Information Fields**
  - Info Field 1: Pointer to event flags group.
  - Info Field 2: Event flags to set (or clear).
  - Info Field 3: AND or OR event flag option.
  - Info Field 4: Number of threads suspended on event flag group.

**Event Flags Set Notify**

- **Icon**: tx_event_flags_set_notify
- **Description**: This event represents registering a notification callback for any event flag set operation on an existing event flags group via `tx_event_flags_set_notify`.
- **Information Fields**
  - Info Field 1: Pointer to event flags group.
  - Info Field 2: Not used.
  - Info Field 3: Not used.
  - Info Field 4: Not used.

**Interrupt Control**

- **Icon**: tx_interrupt_control
- **Description**: This event represents changing the interrupt lockout posture of the processor via `tx_interrupt_control`.
- **Information Fields**
  - Info Field 1: New interrupt posture.
  - Info Field 2: Stack pointer at time of call.
  - Info Field 3: Not used.
  - Info Field 4: Not used.

**Mutex Create**

- **Icon**: tx_mutex_create
- **Description**: This event represents creating a mutex via `tx_mutex_create`.
- **Information Fields**
  - Info Field 1: Pointer to mutex control block.
  - Info Field 2: Priority inheritance option (TX_INHERIT or TX_NO_INHERIT).
  - Info Field 3: Stack pointer at time of call.
  - Info Field 4: Not used.

**Mutex Delete**

- **Icon**: tx_mutex_delete
- **Description**: This event represents deleting a mutex via `tx_mutex_delete`.
- **Information Fields**
  - Info Field 1: Pointer to mutex.
  - Info Field 2: Stack pointer at time of call.
  - Info Field 3: Not used.
  - Info Field 4: Not used.

**Mutex Get**

- **Icon**: tx_mutex_get
- **Description**: This event represents obtaining a mutex via `tx_mutex_get`.
- **Information Fields**
  - Info Field 1: Pointer to mutex.
  - Info Field 2: The wait option supplied to the `tx_mutex_get` call.
  - Info Field 3: Pointer to thread that owns the mutex (NULL implies the mutex is not owned).
  - Info Field 4: Number of times the owning thread has called `tx_mutex_get`.
Mutex Information Get

Icon: tx_mutex_info_get

Description: This event represents retrieving mutex information via tx_mutex_info_get.

Information Fields:
- Info Field 1: Pointer to mutex.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Mutex Performance Information Get

Icon: tx_mutex_performance_info_get

Description: This event represents retrieving mutex performance information via tx_mutex_performance_info_get.

Information Fields:
- Info Field 1: Pointer to mutex.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Mutex Performance System Info Get

Icon: tx_mutex_performance_system_info_get

Description: This event represents retrieving mutex system performance information via tx_mutex_performance_system_info_get.

Information Fields:
- Info Field 1: Not used.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Mutex Prioritize

Icon: tx_mutex_prioritize

Description: This event represents prioritizing the mutex's suspension list via tx_mutex_prioritize.

Information Fields:
- Info Field 1: Pointer to corresponding mutex.
- Info Field 2: Number of threads currently suspended on the mutex.
- Info Field 3: Stack pointer at time of call.
- Info Field 4: Not used.

Mutex Put

Icon: tx_mutex_put

Description: This event represents releasing a previously owned mutex via tx_mutex_put.

Information Fields:
- Info Field 1: Pointer to corresponding mutex.
- Info Field 2: Pointer of thread owning the mutex.
- Info Field 3: Number of outstanding mutex get requests.
- Info Field 4: Stack pointer at time of call.

Queue Create

Icon: tx_queue_create

Description: This event represents creating a message queue via tx_queue_create.

Information Fields:
- Info Field 1: Pointer to queue control block.
- Info Field 2: Size of message – in terms of 32-bit words.
- Info Field 3: Pointer to start of queue memory area.
- Info Field 4: Number of bytes in the queue memory area.
<table>
<thead>
<tr>
<th>Queue Delete</th>
<th>Queue Flush</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents deleting a queue via tx_queue_delete.</td>
<td>This event represents flushing (clearing all queue contents) of a queue via tx_queue_flush.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Pointer to queue.</td>
<td>Info Field 1: Pointer to queue.</td>
</tr>
<tr>
<td>Info Field 2: Stack pointer at time of call.</td>
<td>Info Field 2: Stack pointer at time of call.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue Front Send</th>
<th>Queue Information Get</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents sending a message to the front of a queue via tx_queue_front_send.</td>
<td>This event represents getting information about a queue via tx_queue_info_get.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Pointer to queue.</td>
<td>Info Field 1: Pointer to queue.</td>
</tr>
<tr>
<td>Info Field 2: Pointer to start of message.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Wait option supplied to the tx_queue_front_send call.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Number of messages already enqueued.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue Performance Info Get</th>
<th>Queue Performance System Info Get</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents getting performance information about a queue via tx_queue_performance_info_get.</td>
<td>This event represents getting system performance information about all the queues in the system.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Pointer to queue.</td>
<td>Info Field 1: Not used.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Event Name</td>
<td>Icon</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| Queue Prioritize          | ![Queue Prioritize Icon](image1.png) | This event represents prioritizing the queue's suspension list via tx_queue_prioritize. | Info Field 1: Pointer to corresponding queue.  
Info Field 2: Number of threads currently suspended on the queue.  
Info Field 3: Stack pointer at time of call.  
Info Field 4: Not used.                                                                                       |
| Queue Receive             | ![Queue Receive Icon](image2.png)   | This event represents receiving a message from a queue via tx_queue_receive.     | Info Field 1: Pointer to queue.  
Info Field 2: Pointer to destination for message.  
Info Field 3: Wait option supplied to the call.  
Info Field 4: Number of messages currently queued.                                                            |
| Queue Send                | ![Queue Send Icon](image3.png)      | This event represents sending a message to a queue via tx_queue_send.          | Info Field 1: Pointer to queue.  
Info Field 2: Pointer to message.  
Info Field 3: Wait option supplied to the call.  
Info Field 4: Number of messages currently queued.                                                            |
| Queue Send Notify         | ![Queue Send Notify Icon](image4.png) | This event represents registering a callback via tx_queue_send_notify which is called whenever a message is sent to a queue. | Info Field 1: Pointer to queue.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used.                                                                                       |
| Semaphore Ceiling Put    | ![Semaphore Ceiling Put Icon](image5.png) | This event represents putting to a semaphore via tx_semaphore_ceiling_put. This differs from tx_semaphore_put in that the maximum value of the semaphore is examined such that the put operation is not allowed to exceed the maximum value or ceiling. | Info Field 1: Pointer to semaphore.  
Info Field 2: Current semaphore count.  
Info Field 3: Number of threads suspended on the semaphore.  
Info Field 4: Ceiling limit supplied to the call.                                                             |
| Semaphore Create          | ![Semaphore Create Icon](image6.png) | This event represents creating a semaphore via tx_semaphore_create.            | Info Field 1: Pointer to semaphore control block.  
Info Field 2: Initial semaphore count.  
Info Field 3: Stack pointer at time of call.  
Info Field 4: Not used.                                                                                       |
<table>
<thead>
<tr>
<th>Event</th>
<th>Function</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semaphore Delete</td>
<td>tx_semaphore_delete</td>
<td>This event represents deleting a semaphore via tx_semaphore_delete.</td>
<td>Info Field 1: Pointer to semaphore. Info Field 2: Stack pointer at time of call. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Semaphore Get</td>
<td>tx_semaphore_get</td>
<td>This event represents obtaining a semaphore via tx_semaphore_get.</td>
<td>Info Field 1: Pointer to semaphore. Info Field 2: Wait option supplied to the call. Info Field 3: Current semaphore count. Info Field 4: Stack pointer at time of call.</td>
</tr>
<tr>
<td>Semaphore Information Get</td>
<td>tx_semaphore_info_get</td>
<td>This event represents obtaining information about a semaphore via tx_semaphore_info_get.</td>
<td>Info Field 1: Pointer to semaphore. Info Field 2: Not used. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Semaphore Performance System Info</td>
<td>tx_semaphore_performance_system_info_get</td>
<td>This event represents obtaining performance information about all semaphores in the system via tx_semaphore_performance_system_info_get.</td>
<td>Info Field 1: Not used. Info Field 2: Not used. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Semaphore Prioritize</td>
<td>tx_semaphore_prioritize</td>
<td>This event represents prioritizing the semaphore’s suspension list via tx_semaphore_prioritize.</td>
<td>Info Field 1: Pointer to corresponding semaphore. Info Field 2: Number of threads currently suspended on the semaphore. Info Field 3: Stack pointer at time of call. Info Field 4: Not used.</td>
</tr>
</tbody>
</table>
### Semaphore Put

**Icon**

**tx_semaphore_put**

**Description**
This event represents releasing a semaphore instance via `tx_semaphore_put`.

**Information Fields**
- Info Field 1: Pointer to corresponding semaphore.
- Info Field 2: Current semaphore count.
- Info Field 3: Number of threads suspended on the semaphore.
- Info Field 4: Stack pointer at time of call.

### Semaphore Put Notify

**Icon**

**tx_semaphore_put_notify**

**Description**
This event represents registering a callback via `tx_semaphore_put_notify` that is called whenever a semaphore instance is put.

**Information Fields**
- Info Field 1: Pointer to semaphore.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Thread Create

**Icon**

**tx_thread_create**

**Description**
This event represents creating a thread via `tx_thread_create`.

**Information Fields**
- Info Field 1: Pointer to thread control block.
- Info Field 2: Priority of thread.
- Info Field 3: Stack pointer for thread.
- Info Field 4: Size of stack in bytes.

### Thread Delete

**Icon**

**tx_thread_delete**

**Description**
This event represents deleting a thread via `tx_thread_delete`.

**Information Fields**
- Info Field 1: Pointer to thread.
- Info Field 2: Stack pointer at time of call.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Thread Entry/Exit Notify

**Icon**

**tx_entry_exit_notify**

**Description**
This event represents registering a callback via `tx_entry_exit_notify` that is called whenever a thread is entered or exits.

**Information Fields**
- Info Field 1: Pointer to thread.
- Info Field 2: Thread state at time of the registration.
- Info Field 3: Pointer to stack at time of call.
- Info Field 4: Not used.

### Thread Identify

**Icon**

**tx_thread_identify**

**Description**
This event represents getting the current thread pointer via `tx_thread_identify`.

**Information Fields**
- Info Field 1: Not used.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.
## ThreadX Trace Events

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Function</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Thread Information Get | tx_thread_info_get | This event represents getting information about the specified thread via tx_thread_info_get. | Info Field 1: Pointer to thread.  
Info Field 2: State of thread at time of call.  
Info Field 3: Not used.  
Info Field 4: Not used.  |
| Thread Performance Information Get | tx_thread_performance_info_get | This event represents getting performance information about the specified thread via tx_thread_performance_info_get. | Info Field 1: Pointer to thread.  
Info Field 2: State of thread at time of call.  
Info Field 3: Not used.  
Info Field 4: Not used.  |
| Thread Performance System Info Get | tx_thread_performance_system_info_get | This event represents getting performance information about all threads via tx_thread_performance_system_info_get. | Info Field 1: Not used.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used.  |
| Thread Preemption Change | tx_thread_preemption_change | This event represents changing a thread’s preemption-threshold via tx_thread_preemption_change. | Info Field 1: Pointer to thread.  
Info Field 2: New preemption-threshold.  
Info Field 3: Previous preemption-threshold.  
Info Field 4: Thread’s state at time of call.  |
| Thread Priority Change | tx_thread_priority_change | This event represents changing a thread’s priority via tx_thread_priority_change. | Info Field 1: Pointer to thread.  
Info Field 2: New priority.  
Info Field 3: Previous priority.  
Info Field 4: Thread’s state at time of call.  |
| Thread Relinquish | tx_thread_relinquish | This event represents relinquishing the processor from a thread via tx_thread_relinquish. | Info Field 1: Stack pointer at time of call.  
Info Field 2: Pointer to the next thread to execute.  
Info Field 3: Not used.  
Info Field 4: Not used.  |
<table>
<thead>
<tr>
<th>Event</th>
<th>Code</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Thread Reset    | tx_thread_reset    | ![RS] | This event represents resetting a completed or terminated thread via tx_thread_reset. | Info Field 1: Pointer to thread.  
                      |                    |      |                                                                            | Info Field 2: Thread’s state at time of call.  
                      |                    |      |                                                                            | Info Field 3: Not used.  
                      |                    |      |                                                                            | Info Field 4: Not used. |
| Thread Resume   | tx_thread_resume   | ![TR] | This event represents resuming a suspended thread via tx_thread_resume.       | Info Field 1: Pointer to thread.  
                      |                    |      |                                                                            | Info Field 2: Thread’s state at time of call.  
                      |                    |      |                                                                            | Info Field 3: Stack pointer at time of call.  
                      |                    |      |                                                                            | Info Field 4: Not used. |
| Thread Sleep    | tx_thread_sleep    | ![T2] | This event represents suspending the current thread for a specified number of timer ticks via tx_thread_sleep. | Info Field 1: Number of ticks to suspend for.  
                      |                    |      |                                                                            | Info Field 2: Thread’s state at time of call.  
                      |                    |      |                                                                            | Info Field 3: Stack pointer at time of call.  
                      |                    |      |                                                                            | Info Field 4: Not used. |
| Thread Stack Error Notify | tx_thread_stack_error_notify | ![SE] | This event represents registering a thread stack error notification routine via tx_thread_stack_error_notify_event. | Info Field 1: Not used.  
                      |                    |      |                                                                            | Info Field 2: Not used.  
                      |                    |      |                                                                            | Info Field 3: Not used.  
                      |                    |      |                                                                            | Info Field 4: Not used. |
| Thread Suspend  | tx_thread_suspend  | ![TS] | This event represents suspending a thread via tx_thread_suspend.             | Info Field 1: Pointer to thread to suspend.  
                      |                    |      |                                                                            | Info Field 2: Thread’s state at time of call.  
                      |                    |      |                                                                            | Info Field 3: Stack pointer at time of call.  
                      |                    |      |                                                                            | Info Field 4: Not used. |
| Thread Terminate | tx_thread_terminate | ![TT] | This event represents terminating a thread via tx_thread_terminate.          | Info Field 1: Pointer to thread to terminate.  
                      |                    |      |                                                                            | Info Field 2: Thread’s state at time of call.  
                      |                    |      |                                                                            | Info Field 3: Stack pointer at time of call.  
<pre><code>                  |                    |      |                                                                            | Info Field 4: Not used. |
</code></pre>
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Thread Time-Slice Change   | C    | This event represents changing a thread’s time-slice via `tx_thread_time_slice_change`. | Info Field 1: Pointer to thread.  
Info Field 2: New time-slice.  
Info Field 3: Previous time-slice.  
Info Field 4: Not used. |
| Thread Wait Abort          | W    | This event represents aborting a thread’s suspension via `tx_thread_wait_abort`. | Info Field 1: Pointer to thread.  
Info Field 2: Thread’s state at time of call.  
Info Field 3: Stack pointer at time of call.  
Info Field 4: Not used. |
| Time Get                   | T    | This event represents getting the current number of timer ticks via `tx_time_get`. | Info Field 1: Current number of timer ticks.  
Info Field 2: Stack pointer at time of call.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Time Set                   | S    | This event represents setting the current number of timer ticks via `tx_time_set`. | Info Field 1: New number of timer ticks.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Timer Activate             | A    | This event represents activating the specified timer via `tx_timer_activate`. | Info Field 1: Pointer to timer.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Timer Change               | C    | This event represents changing the specified timer via `tx_timer_change`. | Info Field 1: Pointer to timer.  
Info Field 2: Initial expiration ticks.  
Info Field 3: Reschedule expiration ticks.  
Info Field 4: Not used. |
## Timer Create

**tx_timer_create**

### Icon
![Icon]

### Description
This event represents creating a timer via `tx_timer_create`.

### Information Fields
- Info Field 1: Pointer to timer control block.
- Info Field 2: Initial expiration ticks.
- Info Field 3: Reschedule expiration ticks.
- Info Field 4: Automatic enable value—either `TX_AUTO_ACTIVATE` (1) or `TX_NO_ACTIVATE` (0).

## Timer Deactivate

**tx_timer_deactivate**

### Icon
![Icon]

### Description
This event represents deactivating a timer via `tx_timer_deactivate`.

### Information Fields
- Info Field 1: Pointer to timer.
- Info Field 2: Stack pointer at time of call.
- Info Field 3: Not used.
- Info Field 4: Not used.

## Timer Delete

**tx_timer_delete**

### Icon
![Icon]

### Description
This event represents deleting a timer via `tx_timer_delete`.

### Information Fields
- Info Field 1: Pointer to timer.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

## Timer Information Get

**tx_timer_info_get**

### Icon
![Icon]

### Description
This event represents getting timer information via `tx_timer_info_get`.

### Information Fields
- Info Field 1: Pointer to timer.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

## Timer Performance Information Get

**tx_timer_performance_info_get**

### Icon
![Icon]

### Description
This event represents getting timer performance information via `tx_timer_performance_info_get`.

### Information Fields
- Info Field 1: Pointer to timer.
- Info Field 2: Stack pointer at time of call.
- Info Field 3: Not used.
- Info Field 4: Not used.

## Timer System Performance Info Get

**tx_timer_performance_system_info_get**

### Icon
![Icon]

### Description
This event represents getting all timer performance information via `tx_timer_performance_system_info_get`.

### Information Fields
- Info Field 1: Not used.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.
This chapter contains a description of the FileX events.

- List of Events and Icons 118
- Event Descriptions 121
List of Events and Icons

The following is a list of FileX events displayed by TraceX. The following describes each event:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Icon" /></td>
<td>Internal Logical Sector Cache Miss</td>
</tr>
<tr>
<td><img src="image2" alt="Icon" /></td>
<td>Internal Directory Cache Miss</td>
</tr>
<tr>
<td><img src="image3" alt="Icon" /></td>
<td>Internal Media Flush</td>
</tr>
<tr>
<td><img src="image4" alt="Icon" /></td>
<td>Internal Directory Entry Read</td>
</tr>
<tr>
<td><img src="image5" alt="Icon" /></td>
<td>Internal Directory Entry Write</td>
</tr>
<tr>
<td><img src="image6" alt="Icon" /></td>
<td>Internal I/O Driver Read</td>
</tr>
<tr>
<td><img src="image7" alt="Icon" /></td>
<td>Internal I/O Driver Write</td>
</tr>
<tr>
<td><img src="image8" alt="Icon" /></td>
<td>Internal I/O Driver Flush</td>
</tr>
<tr>
<td><img src="image9" alt="Icon" /></td>
<td>Internal I/O Driver Abort</td>
</tr>
<tr>
<td><img src="image10" alt="Icon" /></td>
<td>Internal I/O Driver Initialize</td>
</tr>
<tr>
<td><img src="image11" alt="Icon" /></td>
<td>Internal I/O Driver Boot Read</td>
</tr>
<tr>
<td><img src="image12" alt="Icon" /></td>
<td>Internal I/O Driver Release Sectors</td>
</tr>
<tr>
<td><img src="image13" alt="Icon" /></td>
<td>Internal I/O Driver Boot Write</td>
</tr>
<tr>
<td><img src="image14" alt="Icon" /></td>
<td>Internal I/O Driver Driver Un-initialize</td>
</tr>
<tr>
<td><img src="image15" alt="Icon" /></td>
<td>Directory Attributes Read (fx_directory_attributes_read)</td>
</tr>
<tr>
<td><img src="image16" alt="Icon" /></td>
<td>Directory Attributes Set (fx_directory_attributes_set)</td>
</tr>
<tr>
<td><img src="image17" alt="Icon" /></td>
<td>Directory Create (fx_directory_create)</td>
</tr>
<tr>
<td><img src="image18" alt="Icon" /></td>
<td>Directory Default Get (fx_directory_default_get)</td>
</tr>
<tr>
<td><img src="image19" alt="Icon" /></td>
<td>Directory Default Set (fx_directory_default_set)</td>
</tr>
</tbody>
</table>
Directory Delete (fx_directory_delete)
Directory First Entry Find (fx_directory_first_entry_find)
Directory First Full Entry Find (fx_directory_first_full_entry_find)
Directory Information Get (fx_directory_information_get)
Directory Local Path Clear (fx_directory_local_path_clear)
Directory Local Path Get (fx_directory_local_path_get)
Directory Local Path Restore (fx_directory_local_path_restore)
Directory Local Path Set (fx_directory_local_path_set)
Directory Long Name Get (fx_directory_long_name_get)
Directory Name Test (fx_directory_name_test)
Directory Next Entry Find (fx_directory_next_entry_find)
Directory Next Full Entry Find (fx_directory_next_full_entry_find)
Directory Rename (fx_directory_rename)
Directory Short Name Get (fx_directory_short_name_get)
File Allocate (fx_file_allocate)
File Attributes Read (fx_file_attributes_read)
File Attributes Set (fx_file_attributes_set)
File Best Effort Allocate (fx_file_best_effort_allocate)
File Close (fx_file_close)
File Create (fx_file_create)
File Date Time Set (fx_file_date_time_set)
File Delete (fx_file_delete)
File Open \((\text{fx\_file\_open})\)

File Read \((\text{fx\_file\_read})\)

File Relative Seek \((\text{fx\_file\_relative\_seek})\)

File Rename \((\text{fx\_file\_rename})\)

File Seek \((\text{fx\_file\_seek})\)

File Truncate \((\text{fx\_file\_truncate})\)

File Truncate Release \((\text{fx\_file\_truncate\_release})\)

File Write \((\text{fx\_file\_write})\)

Media Abort \((\text{fx\_media\_abort})\)

Media Cache Invalidate \((\text{fx\_media\_cache\_invalidate})\)

Media Check \((\text{fx\_media\_check})\)

Media Close \((\text{fx\_media\_close})\)

Media Flush \((\text{fx\_media\_flush})\)

Media Format \((\text{fx\_media\_format})\)

Media Open \((\text{fx\_media\_open})\)

Media Read \((\text{fx\_media\_read})\)

Media Space Available \((\text{fx\_media\_space\_available})\)

Media Volume Get \((\text{fx\_media\_volume\_get})\)

Media Volume Set \((\text{fx\_media\_volume\_set})\)

Media Write \((\text{fx\_media\_write})\)

System Date Get \((\text{fx\_system\_date\_get})\)

System Date Set \((\text{fx\_system\_date\_set})\)
Event Descriptions

The following describes each individual event.
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| **Internal Directory Cache Miss**| ![Icon](image1) | This event represents an internal FileX directory cache miss.               | Info Field 1: Pointer to the media.  
Info Field 2: Total misses.  
Info Field 3: Not used.  
Info Field 4: Not used.                                                                 |
| **Internal Directory Entry Read** | ![Icon](image2) | This event represents an internal FileX directory entry read event.          | Info Field 1: Pointer to the media.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used.                                                                 |
| **Internal Directory Entry Write**| ![Icon](image3) | This event represents an internal FileX directory entry write event.         | Info Field 1: Pointer to the media.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used.                                                                 |
| **Internal I/O Driver Abort**     | ![Icon](image4) | This event represents an internal FileX I/O driver abort event.             | Info Field 1: Pointer to the media.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used.                                                                 |
| **Internal I/O Driver Boot Sector Read** | ![Icon](image5) | This event represents an internal FileX I/O driver boot sector read event. | Info Field 1: Pointer to the media.  
Info Field 2: Buffer pointer.  
Info Field 3: Not used.  
Info Field 4: Not used.                                                                 |
| **Internal I/O Driver Boot Sector Write** | ![Icon](image6) | This event represents an internal FileX I/O driver boot sector write event. | Info Field 1: Pointer to the media.  
Info Field 2: Buffer pointer.  
Info Field 3: Not used.  
Info Field 4: Not used.                                                                 |
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal I/O Driver Flush</td>
<td>This event represents an internal FileX I/O driver flush event.</td>
<td>Info Field 1: Pointer to the media.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Internal I/O Driver Un-initialize</td>
<td>This event represents an internal FileX I/O driver un-initialize event.</td>
<td>Info Field 1: Pointer to the media.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Internal I/O Driver Initialize</td>
<td>This event represents an internal FileX I/O driver initialize event.</td>
<td>Info Field 1: Pointer to the media.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Internal I/O Driver Read</td>
<td>This event represents an internal FileX I/O driver read event.</td>
<td>Info Field 1: Pointer to the media.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Sector.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Number of sectors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Buffer pointer.</td>
</tr>
<tr>
<td>Internal I/O Driver Release Sectors</td>
<td>This event represents an internal FileX I/O driver release sectors event.</td>
<td>Info Field 1: Pointer to the media.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Sector.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Number of sectors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Internal I/O Driver Write</td>
<td>This event represents an internal FileX I/O driver write event.</td>
<td>Info Field 1: Pointer to the media.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Sector.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Number of sectors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Buffer pointer.</td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
<td>Information Fields</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **Internal Logical Sector Cache Miss** | This event represents an internal FileX logical sector cache miss.          | Info Field 1: Pointer to the media.  
Info Field 2: Sector.  
Info Field 3: Total misses.  
Info Field 4: Cache size. |
| **Internal Media Flush**     | This event represents an internal FileX media flush.                        | Info Field 1: Pointer to the media.  
Info Field 2: Number of dirty sectors.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **File Attributes Read**     | This event represents a file attributes read event.                         | Info Field 1: Pointer to the media.  
Info Field 2: Attributes bit map:  
Read Only (0x01)  
Hidden (0x02)  
System (0x04)  
Volume (0x08)  
Directory (0x10)  
Archive (0x20)  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **File Attributes Set**      | This event represents a file attributes set event.                          | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to file name.  
Info Field 3: Attributes bit map:  
Read Only (0x01)  
Hidden (0x02)  
System (0x04)  
Archive (0x20)  
Info Field 4: Not used. |
| **File Best Effort Allocate** | This event represents a file best effort allocate event.                    | Info Field 1: Pointer to the file.  
Info Field 2: Requested size.  
Info Field 3: Actual size allocated.  
Info Field 4: Not used. |
| **File Close**                | This event represents a file close event.                                   | Info Field 1: Pointer to the file.  
Info Field 2: File size.  
Info Field 3: Not used.  
Info Field 4: Not used. |
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Function Name</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Directory Default Set      | fx_directory_default_set | Represents a directory default set event.                                    | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to new default path name.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Directory Delete           | fx_directory_delete    | Represents a directory delete event.                                         | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to directory name.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Directory First Entry Find | fx_directory_first_entry_find | Represents a directory first entry find event.                               | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to directory name.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Directory First Full Entry Find | fx_directory_first_full_entry_find | Represents a directory first full entry find event.                          | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to directory name.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Directory Information Get  | fx_directory_information_get | Represents a directory information get event.                               | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to directory name.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Directory Local Path Clear | fx_directory_local_path_clear | Represents a directory local path clear event.                               | Info Field 1: Pointer to the media.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
### Directory Local Path Get

**Icon**: L

**Description**
This event represents a directory local path get event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to return path name.
- Info Field 3: Not used.
- Info Field 4: Not used.

---

### Directory Local Path Restore

**Icon**: R

**Description**
This event represents a directory local path restore event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to local path structure.
- Info Field 3: Not used.
- Info Field 4: Not used.

---

### Directory Local Path Set

**Icon**: S

**Description**
This event represents a directory local path set event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to local path structure.
- Info Field 3: Pointer to new path name.
- Info Field 4: Not used.

---

### Directory Long Name Get

**Icon**: N

**Description**
This event represents a directory long name get event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to short file name.
- Info Field 3: Pointer to long file name.
- Info Field 4: Not used.

---

### Directory Name Test

**Icon**: T

**Description**
This event represents a directory name test event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to directory name.
- Info Field 3: Not used.
- Info Field 4: Not used.

---

### Directory Next Entry Find

**Icon**: E

**Description**
This event represents a directory next entry find event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to directory name.
- Info Field 3: Not used.
- Info Field 4: Not used.
### Directory Next Full Entry Find

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| N    | This event represents a directory next full entry find event. | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to directory name.  
Info Field 3: Not used.  
Info Field 4: Not used. |

### Directory Rename

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| D    | This event represents a directory rename event. | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to old directory name.  
Info Field 3: Pointer to new directory name.  
Info Field 4: Not used. |

### Directory Short Name Get

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| S    | This event represents a directory short name get event. | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to long file name.  
Info Field 3: Pointer to short file name.  
Info Field 4: Not used. |

### File Allocate

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| F    | This event represents a file allocate event. | Info Field 1: Pointer to the file.  
Info Field 2: Requested size.  
Info Field 3: Current size.  
Info Field 4: New size. |

### File Create

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| C    | This event represents a file create event. | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to file name.  
Info Field 3: Not used.  
Info Field 4: Not used. |

### File Date Time Set

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| T    | This event represents a file date/time set event. | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to file name.  
Info Field 3: Not used.  
Info Field 4: Not used.  
Info Field 5: Year.  
Info Field 6: Month. |
<table>
<thead>
<tr>
<th>Event</th>
<th>Function</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Delete</strong></td>
<td><code>fx_file_delete</code></td>
<td>This event represents a file delete event.</td>
<td>Info Field 1: Pointer to the media.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Pointer to file name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>File Open</strong></td>
<td><code>fx_file_open</code></td>
<td>This event represents a file open event.</td>
<td>Info Field 1: Pointer to the media.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Pointer to the file control block.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Pointer to file name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Open type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Open for Read (0x00)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Open for Write (0x01)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fast Open for Read (0x02)</td>
</tr>
<tr>
<td><strong>File Read</strong></td>
<td><code>fx_file_read</code></td>
<td>This event represents a file read event.</td>
<td>Info Field 1: Pointer to the file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Buffer pointer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Request size.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Actual size read.</td>
</tr>
<tr>
<td><strong>File Relative Seek</strong></td>
<td><code>fx_file_relative_seek</code></td>
<td>This event represents a file relative seek event.</td>
<td>Info Field 1: Pointer to the file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Byte offset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Seek from:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>From Beginning (0x00)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>From End (0x01)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Forward (0x02)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Backward (0x03)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Previous offset.</td>
</tr>
<tr>
<td><strong>File Rename</strong></td>
<td><code>fx_file_rename</code></td>
<td>This event represents a file rename event.</td>
<td>Info Field 1: Pointer to the media.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Pointer to old file name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Pointer to new file name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>File Seek</strong></td>
<td><code>fx_file_seek</code></td>
<td>This event represents a file seek event.</td>
<td>Info Field 1: Pointer to the file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Byte offset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Previous offset.</td>
</tr>
</tbody>
</table>
### File Truncate

**Icon**: T

**Description**: This event represents a file truncate event.

**Information Fields**
- Info Field 1: Pointer to the file.
- Info Field 2: Requested size.
- Info Field 3: Previous size.
- Info Field 4: New size.

### File Truncate Release

**Icon**: T

**Description**: This event represents a file truncate release event.

**Information Fields**
- Info Field 1: Pointer to the file.
- Info Field 2: Requested size.
- Info Field 3: Previous size.
- Info Field 4: New size.

### File Write

**Icon**: F

**Description**: This event represents a file write event.

**Information Fields**
- Info Field 1: Pointer to the file.
- Info Field 2: Buffer pointer.
- Info Field 3: Request size.
- Info Field 4: Actual size written.

### Media Abort

**Icon**: A

**Description**: This event represents a media abort event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Media Cache Invalidate

**Icon**: C

**Description**: This event represents a media cache invalidate event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Media Check

**Icon**: C

**Description**: This event represents a media check event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Scratch memory pointer.
- Info Field 3: Scratch memory size.
- Info Field 4: Errors bit map:
  - FAT Chain Error (0x01)
  - Directory Error (0x02)
  - Lost Cluster Error (0x04)
  - File Size Error (0x08)
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Icon</th>
<th>Function Call</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Media Close         | ![Icon](media_close.png) | fx_media_close  | This event represents a media close event.                                | Info Field 1: Pointer to the media.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Media Flush         | ![Icon](media_flush.png) | fx_media_flush  | This event represents a media flush event.                                | Info Field 1: Pointer to the media.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Media Format        | ![Icon](media_format.png) | fx_media_format | This event represents a media format event.                                | Info Field 1: Pointer to the media.  
Info Field 2: Number of root entries.  
Info Field 3: Sectors.  
Info Field 4: Sectors per cluster. |
| Media Open          | ![Icon](media_open.png) | fx_media_open   | This event represents a media open event.                                 | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to media driver entry.  
Info Field 3: Memory pointer.  
Info Field 4: Memory size. |
| Media Read          | ![Icon](media_read.png)  | fx_media_read   | This event represents a media read event.                                 | Info Field 1: Pointer to the media.  
Info Field 2: Logical sector.  
Info Field 3: Buffer pointer.  
Info Field 4: Bytes read. |
| Media Space Available| ![Icon](media_space_available.png) | fx_media_space_available | This event represents a media space available event.                       | Info Field 1: Pointer to the media.  
Info Field 2: Available bytes pointer.  
Info Field 3: Number of free clusters.  
Info Field 4: Not used. |
## Media Volume Get

**Icon** ![Icon](tracex_icon)

**Description**
This event represents a media volume get event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to volume name.
- Info Field 3: Volume source.
- Info Field 4: Not used.

## Media Volume Set

**Icon** ![Icon](tracex_icon)

**Description**
This event represents a media volume set event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to volume name.
- Info Field 3: Not used.
- Info Field 4: Not used.

## Media Write

**Icon** ![Icon](tracex_icon)

**Description**
This event represents a media write event.

**Information Fields**
- Info Field 1: Pointer to the media.
- Info Field 2: Logical sector.
- Info Field 3: Buffer pointer.
- Info Field 4: Bytes written.

## System Date Get

**Icon** ![Icon](tracex_icon)

**Description**
This event represents a system date get event.

**Information Fields**
- Info Field 1: Year.
- Info Field 2: Month.
- Info Field 3: Day.
- Info Field 4: Not used.

## System Date Set

**Icon** ![Icon](tracex_icon)

**Description**
This event represents a system date set event.

**Information Fields**
- Info Field 1: Year.
- Info Field 2: Month.
- Info Field 3: Day.
- Info Field 4: Not used.

## System Initialize

**Icon** ![Icon](tracex_icon)

**Description**
This event represents a system initialize event.

**Information Fields**
- Info Field 1: Not used.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Time Get</td>
<td>fx_system_time_get</td>
<td><img src="system_time_get_icon.png" alt="Image" /></td>
<td>This event represents a system time get event.</td>
<td>Info Field 1: Hour. Info Field 2: Minute. Info Field 3: Second. Info Field 4: Not used.</td>
</tr>
<tr>
<td>System Time Set</td>
<td>fx_system_time_set</td>
<td><img src="system_time_set_icon.png" alt="Image" /></td>
<td>This event represents a system time set event.</td>
<td>Info Field 1: Hour. Info Field 2: Minute. Info Field 3: Second. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Unicode Directory Create</td>
<td>fx_unicode_directory_create</td>
<td><img src="unicode_directory_create_icon.png" alt="Image" /></td>
<td>This event represents a Unicode directory create event.</td>
<td>Info Field 1: Pointer to the media. Info Field 2: Pointer to Unicode name. Info Field 3: Size of Unicode name. Info Field 4: Pointer to short name.</td>
</tr>
<tr>
<td>Unicode Directory Rename</td>
<td>fx_unicode_directory_rename</td>
<td><img src="unicode_directory_rename_icon.png" alt="Image" /></td>
<td>This event represents a Unicode directory rename event.</td>
<td>Info Field 1: Pointer to the media. Info Field 2: Pointer to Unicode name. Info Field 3: Size of Unicode name. Info Field 4: Pointer to short name.</td>
</tr>
<tr>
<td>Unicode File Create</td>
<td>fx_unicode_file_create</td>
<td><img src="unicode_file_create_icon.png" alt="Image" /></td>
<td>This event represents a Unicode file create event.</td>
<td>Info Field 1: Pointer to the media. Info Field 2: Pointer to the Unicode name. Info Field 3: Size of Unicode name. Info Field 4: Pointer to short name.</td>
</tr>
<tr>
<td>Unicode File Rename</td>
<td>fx_unicode_file_rename</td>
<td><img src="unicode_file_rename_icon.png" alt="Image" /></td>
<td>This event represents a Unicode file rename event.</td>
<td>Info Field 1: Pointer to the media. Info Field 2: Pointer to Unicode name. Info Field 3: Size of Unicode name. Info Field 4: Pointer to short name.</td>
</tr>
<tr>
<td>Event Name</td>
<td>Function Call</td>
<td>Icon</td>
<td>Description</td>
<td>Information Fields</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------</td>
<td>------</td>
<td>----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Unicode Length Get         | `fx_unicode_length_get`| ![Icon](image) | This event represents a Unicode length get event.                           | Info Field 1: Pointer to the Unicode name.  
Info Field 2: Length.  
Info Field 3: Not used.  
Info Field 4: Not used. |
|                            |                        |      |                                                                            |                                                                                     |
| Unicode Name Get           | `fx_unicode_name_get`  | ![Icon](image) | This event represents a Unicode name get event.                             | Info Field 1: Pointer to the media.  
Info Field 2: Source short name.  
Info Field 3: Destination Unicode name pointer.  
Info Field 4: Destination Unicode name length. |
|                            |                        |      |                                                                            |                                                                                     |
| Unicode Short Name Get     | `fx_unicode_short_name_get` | ![Icon](image) | This event represents a Unicode short name get event.                      | Info Field 1: Pointer to the media.  
Info Field 2: Pointer to source Unicode name.  
Info Field 3: Length of Unicode name.  
Info Field 4: Pointer to short name. |
This chapter contains a description of the NetX events.

- List of Events and Icons 136
- Event Descriptions 143
## List of Events and Icons

The following is a list of NetX events displayed by TraceX.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Internal ARP Request Receive</td>
</tr>
<tr>
<td></td>
<td>Internal ARP Request Send</td>
</tr>
<tr>
<td></td>
<td>Internal ARP Response Receive</td>
</tr>
<tr>
<td></td>
<td>Internal ARP Response Send</td>
</tr>
<tr>
<td></td>
<td>Internal ICMP Receive</td>
</tr>
<tr>
<td></td>
<td>Internal ICMP Send</td>
</tr>
<tr>
<td></td>
<td>Internal NetX IGMP Receive</td>
</tr>
<tr>
<td></td>
<td>Internal IP Receive</td>
</tr>
<tr>
<td></td>
<td>Internal IP Send</td>
</tr>
<tr>
<td></td>
<td>Internal TCP Data Receive</td>
</tr>
<tr>
<td></td>
<td>Internal TCP Data Send</td>
</tr>
<tr>
<td></td>
<td>Internal TCP FIN Receive</td>
</tr>
<tr>
<td></td>
<td>Internal TCP FIN Send</td>
</tr>
<tr>
<td></td>
<td>Internal TCP RST Receive</td>
</tr>
<tr>
<td></td>
<td>Internal TCP RST Send</td>
</tr>
<tr>
<td></td>
<td>Internal TCP SYN Receive</td>
</tr>
<tr>
<td></td>
<td>Internal TCP SYN Send</td>
</tr>
<tr>
<td></td>
<td>Internal UDP Receive</td>
</tr>
<tr>
<td></td>
<td>Internal UDP Send</td>
</tr>
</tbody>
</table>
Internal RARP Receive
Internal RARP Send
Internal TCP Retry
Internal TCP State Change
Internal I/O Driver Packet Send
Internal I/O Driver Initialize
Internal I/O Driver Link Enable
Internal I/O Driver Link Disable
Internal I/O Driver Packet Broadcast
Internal I/O Driver ARP Send
Internal I/O Driver ARP Response Send
Internal I/O Driver RARP Send
Internal I/O Driver Multicast Join
Internal I/O Driver Multicast Leave
Internal I/O Driver Get Status
Internal I/O Driver Get Speed
Internal I/O Driver Get Duplex Type
Internal I/O Driver Get Error Count
Internal I/O Driver Get RX Count
Internal I/O Driver Get TX Count
Internal I/O Driver Get Allocation Errors
Internal I/O Driver Un-initialize
Internal I/O Driver Deferred Processing

ARP Dynamic Entries Invalidate
   (nx_arp_dynamic_entries_invalidate)

ARP Dynamic Entry Set (nx_arp_dynamic_entry_set)

ARP Enable (nx_arp_enable)

ARP Gratuitous Send (nx_arp_gratuitous_send)

ARP Hardware Address Find (nx_arp.hardware_address_find)

ARP Information Get (nx_arp_info_get)

ARP IP Address Find (nx_arp_ip_address_find)

ARP Static Entries Delete (nx_arp_static_entries_delete)

ARP Static Entry Create (nx_arp_static_entry_create)

ARP Static Entry Delete (nx_arp_static_entry_delete)

Duo Cache Entry Delete (nxd.nd_cache_entry_delete)

Duo Cache Entry Set (nxd.nd_cache_entry_set)

Duo Cache Invalidate (nxd.nd_cache_invalidate)

Duo Cache IP Address Find (nxd.nd_cache_ip_address_find)

Duo ICMP Enable (nxd.icmp_enable)

Duo ICMP IPv6 Ping (nxd.icmp.ping)

Duo IP Max Payload Size Find (nx_max.payload.size_find)

Duo IP Raw Packet Send (nxd.ip_raw.packet.send)

Duo IPv6 Default Router Add (nxd_ipv6.default_router_add)

Duo IPv6 Default Router Delete (nxd_ipv6.default_router_delete)

Duo IPv6 Enable (nxd_ipv6.enable)
Duo IPv6 Global Address Get \((nxd_ipv6_global_address_get)\)

Duo IPv6 Global Address Set \((nxd_ipv6_global_address_set)\)

Duo IPv6 Initiate Dad Process \((nxd_ipv6_initiate_dad_process)\)

Duo IPv6 Interface Address Get \((nxd_ipv6_interface_address_get)\)

Duo IPv6 Interface Address Set \((nxd_ipv6_interface_address_set)\)

Duo IPv6 Link Local Address Get \((nxd_ipv6_linklocal_address_get)\)

Duo IPv6 Link Local Address Set \((nxd_ipv6_linklocal_address_set)\)

Duo IPv6 Raw Packet Send \((nxd_ipv6_raw_packet_send)\)

Duo TCP Socket Peer Info Get \((nxd_tcp_socket_peer_info_get)\)

Duo TCP Socket Set Interface \((nxd_tcp_socket_set_interface)\)

Duo UDP Socket Send \((nxd_udp_socket_send)\)

Duo UDP Socket Set Interface \((nxd_udp_socket_set_interface)\)

Duo UDP Source Extract \((nxd_udp_source_extract)\)

ICMP Enable \((nx_icmp_enable)\)

ICMP Information Get \((nx_icmp_info_get)\)

ICMP Ping \((nx_icmp_ping)\)

IGMP Enable \((nx_igmp_enable)\)

IGMP Information Get \((nx_igmp_info_get)\)

IGMP Loopback Disable \((nx_igmp_loopback_disable)\)

IGMP Loopback Enable \((nx_igmp_loopback_enable)\)

IGMP Multicast Join \((nx_igmp_multicast_join)\)

IGMP Multicast Leave \((nx_igmp_multicast_leave)\)
IP Address Change Notify (nx_ip_address_change_notify)
IP Address Get (nx_ip_address_get)
IP Address Set (nx_ip_address_set)
IP Create (nx_ip_create)
IP Delete (nx_ip_delete)
IP Driver Direct Command (nx_ip_driver_direct_command)
IP Forwarding Disable (nx_ip_forwarding_disable)
IP Forwarding Enable (nx_ip_forwarding_enable)
IP Fragment Disable (nx_ip_fragment_disable)
IP Fragment Enable (nx_ip_fragment_enable)
IP Gateway Address Set (nx_ip_gateway_address_set)
IP Information Get (nx_ip_info_get)
IP Interface Attach (nx_ip_interface_attach)
IP Interface Info Get (nx_ip_interface_info_get)
IP Raw Packet Disable (nx_ip_raw_packet_disable)
IP Raw Packet Enable (nx_ip_raw_packet_enable)
IP Raw Packet Receive (nx_ip_raw_packet_receive)
IP Raw Packet Send (nx_ip_raw_packet_send)
IP Status Check (nx_ip_status_check)
IP Static Route Add (nx_ip_static_route_add)
IP Static Route Delete (nx_ip_static_route_delete)
IPSEC Enable (nx_ipsec_enable)
Packet Allocate \((nx\_packet\_allocate)\)

Packet Copy \((nx\_packet\_copy)\)

Packet Data Append \((nx\_packet\_data\_append)\)

Packet Data Extract Offset \((nx\_packet\_data\_extract\_offset)\)

Packet Data Retrieve \((nx\_packet\_data\_retrieve)\)

Packet Length Get \((nx\_packet\_length\_get)\)

Packet Pool Create \((nx\_packet\_pool\_create)\)

Packet Pool Delete \((nx\_packet\_pool\_delete)\)

Packet Pool Information Get \((nx\_packet\_pool\_info\_get)\)

Packet Release \((nx\_packet\_release)\)

Packet Transmit Release \((nx\_packet\_transmit\_release)\)

RARP Disable \((nx\_rarp\_disable)\)

RARP Enable \((nx\_rarp\_enable)\)

RARP Information Get \((nx\_rarp\_info\_get)\)

System Initialize \((nx\_system\_initialize)\)

TCP Client Socket Bind \((nx\_tcp\_client\_socket\_bind)\)

TCP Client Socket Connect \((nx\_tcp\_client\_socket\_connect)\)

TCP Client Socket Port Get \((nx\_tcp\_client\_socket\_port\_get)\)

TCP Client Socket Unbind \((nx\_tcp\_client\_socket\_unbind)\)

TCP Enable \((nx\_tcp\_enable)\)

TCP Free Port Find \((nx\_tcp\_free\_port\_find)\)

TCP Information Get \((nx\_tcp\_info\_get)\)

TCP Server Socket Accept \((nx\_tcp\_server\_socket\_accept)\)
TCP Server Socket Listen \((nx\_tcp\_server\_socket\_listen)\)

TCP Server Socket Relisten \((nx\_tcp\_server\_socket\_relisten)\)

TCP Server Socket Unaccept \((nx\_tcp\_server\_socket\_unaccept)\)

TCP Server Socket Unlisten \((nx\_tcp\_server\_socket\_unlisten)\)

TCP Socket Bytes Available \((nx\_tcp\_socket\_bytes\_available)\)

TCP Socket Create \((nx\_tcp\_socket\_create)\)

TCP Socket Delete \((nx\_tcp\_socket\_delete)\)

TCP Socket Disconnect \((nx\_tcp\_socket\_disconnect)\)

TCP Socket Information Get \((nx\_tcp\_socket\_info\_get)\)

TCP Socket MSS Get \((nx\_tcp\_socket\_mss\_get)\)

TCP Socket MSS Peer Get \((nx\_tcp\_socket\_mss\_peer\_get)\)

TCP Socket MSS Set \((nx\_tcp\_socket\_mss\_set)\)

TCP Socket Peer Info Get \((nx\_tcp\_socket\_peer\_info\_get)\)

TCP Socket Receive \((nx\_tcp\_socket\_receive)\)

TCP Socket Receive Notify \((nx\_tcp\_socket\_receive\_notify)\)

TCP Socket Send \((nx\_tcp\_socket\_send)\)

TCP Socket State Wait \((nx\_tcp\_socket\_state\_wait)\)

TCP Socket Transmit Configure \((nx\_tcp\_socket\_transmit\_configure)\)

TCP Socket Window Update Notify Set
\((nx\_tcp\_socket\_window\_update\_notify\_set)\)

UDP Enable \((nx\_udp\_enable)\)

UDP Free Port Find \((nx\_udp\_free\_port\_find)\)

UDP Information Get \((nx\_udp\_info\_get)\)
UDP Socket Bind (*nx_udp_socket_bind*)

UDP Socket Bytes Available (*nx_udp_socket_bytes_available*)

UDP Socket Checksum Disable (*nx_udp_socket_checksum_disable*)

UDP Socket Checksum Enable (*nx_udp_socket_checksum_enable*)

UDP Socket Create (*nx_udp_socket_create*)

UDP Socket Delete (*nx_udp_socket_delete*)

UDP Socket Information Get (*nx_udp_socket_info_get*)

UDP Socket Interface Set (*nx_udp_socket_interface_set*)

UDP Socket Port Get (*nx_udp_socket_port_get*)

UDP Socket Receive (*nx_udp_socket_receive*)

UDP Socket Receive Notify (*nx_udp_socket_receive_notify*)

UDP Socket Send (*nx_udp_socket_send*)

UDP Source Extract (*nx_udp_source_extract*)

Event Descriptions

The following pages describe the NetX Trace Events.
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal ARP Request Receive</td>
<td>This event represents an internal NetX ARP request receive event.</td>
<td>Info Field 1: Pointer to the IP instance&lt;br_Info Field 2: Source IP address&lt;br_Info Field 3: Pointer to packet&lt;br_Info Field 4: Not used</td>
</tr>
<tr>
<td>Internal ARP Request Send</td>
<td>This event represents an internal NetX ARP request send event.</td>
<td>Info Field 1: Pointer to the IP instance&lt;br_Info Field 2: Destination IP address&lt;br_Info Field 3: Pointer to packet&lt;br_Info Field 4: Not used</td>
</tr>
<tr>
<td>Internal ARP Response Receive</td>
<td>This event represents an internal NetX ARP response receive event.</td>
<td>Info Field 1: Pointer to the IP instance&lt;br_Info Field 2: Source IP address&lt;br_Info Field 3: Pointer to packet&lt;br_Info Field 4: Not used</td>
</tr>
<tr>
<td>Internal ARP Response Send</td>
<td>This event represents an internal NetX ARP response send event.</td>
<td>Info Field 1: Pointer to the IP instance&lt;br_Info Field 2: Destination IP address&lt;br_Info Field 3: Pointer to packet&lt;br_Info Field 4: Not used</td>
</tr>
<tr>
<td>Internal ICMP Receive</td>
<td>This event represents an internal NetX ICMP receive event.</td>
<td>Info Field 1: Pointer to the IP instance&lt;br_Info Field 2: Source IP address&lt;br_Info Field 3: Pointer to packet&lt;br_Info Field 4: Word 0 of ICMP header</td>
</tr>
<tr>
<td>Internal ICMP Send</td>
<td>This event represents an internal NetX ICMP send event.</td>
<td>Info Field 1: Pointer to the IP instance&lt;br_Info Field 2: Destination IP address&lt;br_Info Field 3: Pointer to packet&lt;br_Info Field 4: Word 0 of ICMP header</td>
</tr>
</tbody>
</table>
Internal IGMP Receive

**Icon**

**Description**
This event represents an internal NetX IGMP receive event.

**Information Fields**
- Info Field 1: IP Pointer
- Info Field 2: Source IP address
- Info Field 3: Pointer to packet
- Info Field 4: Word 0 of IGMP header

---

Internal IP Receive

**Icon**

**Description**
This event represents an internal NetX IP receive event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Source IP address
- Info Field 3: Pointer to packet
- Info Field 4: Packet length

---

Internal IP Send

**Icon**

**Description**
This event represents an internal NetX IP send event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Destination IP address
- Info Field 3: Pointer to packet
- Info Field 4: Packet length

---

Internal TCP Data Receive

**Icon**

**Description**
This event represents an internal NetX TCP data receive event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Source IP address
- Info Field 3: Pointer to packet
- Info Field 4: Receive sequence number

---

Internal TCP Data Send

**Icon**

**Description**
This event represents an internal NetX TCP data send event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Pointer to packet
- Info Field 4: Transmit sequence number

---

Internal TCP FIN Receive

**Icon**

**Description**
This event represents an internal NetX TCP FIN receive event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Pointer to packet
- Info Field 4: Receive sequence number
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal TCP FIN Send</td>
<td>This event represents an internal NetX TCP FIN send event.</td>
</tr>
<tr>
<td>Information Fields</td>
<td>- Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td>- Info Field 2: Pointer to socket</td>
</tr>
<tr>
<td></td>
<td>- Info Field 3: Pointer to packet</td>
</tr>
<tr>
<td></td>
<td>- Info Field 4: Transmit sequence number</td>
</tr>
<tr>
<td>Internal TCP RST Receive</td>
<td>This event represents an internal NetX TCP reset receive event.</td>
</tr>
<tr>
<td>Information Fields</td>
<td>- Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td>- Info Field 2: Pointer to socket</td>
</tr>
<tr>
<td></td>
<td>- Info Field 3: Pointer to packet</td>
</tr>
<tr>
<td></td>
<td>- Info Field 4: Receive sequence number</td>
</tr>
<tr>
<td>Internal TCP RST Send</td>
<td>This event represents an internal NetX TCP reset send event.</td>
</tr>
<tr>
<td>Information Fields</td>
<td>- Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td>- Info Field 2: Pointer to socket</td>
</tr>
<tr>
<td></td>
<td>- Info Field 3: Pointer to packet</td>
</tr>
<tr>
<td></td>
<td>- Info Field 4: Transmit sequence number</td>
</tr>
<tr>
<td>Internal TCP SYN Receive</td>
<td>This event represents an internal NetX TCP SYN receive event.</td>
</tr>
<tr>
<td>Information Fields</td>
<td>- Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td>- Info Field 2: Pointer to socket</td>
</tr>
<tr>
<td></td>
<td>- Info Field 3: Pointer to packet</td>
</tr>
<tr>
<td></td>
<td>- Info Field 4: Receive sequence number</td>
</tr>
<tr>
<td>Internal TCP SYN Send</td>
<td>This event represents an internal NetX TCP SYN send event.</td>
</tr>
<tr>
<td>Information Fields</td>
<td>- Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td>- Info Field 2: Pointer to socket</td>
</tr>
<tr>
<td></td>
<td>- Info Field 3: Pointer to packet</td>
</tr>
<tr>
<td></td>
<td>- Info Field 4: Transmit sequence number</td>
</tr>
<tr>
<td>Internal UDP Receive</td>
<td>This event represents an internal NetX UDP receive event.</td>
</tr>
<tr>
<td>Information Fields</td>
<td>- Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td>- Info Field 2: Pointer to socket</td>
</tr>
<tr>
<td></td>
<td>- Info Field 3: Pointer to packet</td>
</tr>
<tr>
<td></td>
<td>- Info Field 4: Word 0 of UDP header</td>
</tr>
<tr>
<td>Event Name</td>
<td>Icon</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
</tr>
</tbody>
</table>
| Internal UDP Send|      | This event represents an internal NetX UDP send event.                       | Info Field 1: Pointer to the IP instance  
|                  |      |                                                                              | Info Field 2: Pointer to socket  
|                  |      |                                                                              | Info Field 3: Pointer to packet  
|                  |      |                                                                              | Info Field 4: Word 0 of UDP header                                           |
| Internal RARP Receive|      | This event represents an internal NetX RARP receive event.                   | Info Field 1: Pointer to the IP instance  
|                  |      |                                                                              | Info Field 2: Target IP address  
|                  |      |                                                                              | Info Field 3: Pointer to packet  
|                  |      |                                                                              | Info Field 4: Word 1 of RARP header                                          |
| Internal RARP Send|      | This event represents an internal NetX RARP send event.                      | Info Field 1: Pointer to the IP instance  
|                  |      |                                                                              | Info Field 2: Target IP address  
|                  |      |                                                                              | Info Field 3: Pointer to packet  
|                  |      |                                                                              | Info Field 4: Word 1 of RARP header                                          |
| Internal TCP Retry|      | This event represents an internal NetX TCP retry event.                      | Info Field 1: Pointer to the IP instance  
|                  |      |                                                                              | Info Field 2: Pointer to socket  
|                  |      |                                                                              | Info Field 3: Pointer to packet  
|                  |      |                                                                              | Info Field 4: Number of retries                                              |
| Internal TCP State Change| | This event represents an internal NetX TCP socket state change event.       | Info Field 1: Pointer to the IP instance  
|                  |      |                                                                              | Info Field 2: Pointer to socket  
|                  |      |                                                                              | Info Field 3: Previous state  
|                  |      |                                                                              | Info Field 4: New state                                                      |
| Internal I/O Driver Packet Send| | This event represents an internal NetX I/O driver packet send event.         | Info Field 1: Pointer to the IP instance  
|                  |      |                                                                              | Info Field 2: Pointer to packet  
|                  |      |                                                                              | Info Field 3: Packet size                                                     |
### Internal I/O Driver Initialize

**Icon**

**Description**
This event represents an internal NetX I/O driver initialize event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

### Internal I/O Driver Link Enable

**Icon**

**Description**
This event represents an internal NetX I/O driver link enable event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

### Internal I/O Driver Link Disable

**Icon**

**Description**
This event represents an internal NetX I/O driver link disable event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

### Internal I/O Driver Packet Broadcast

**Icon**

**Description**
This event represents an internal NetX I/O driver packet broadcast event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Pointer to packet
- Info Field 3: Packet size
- Info Field 4: Not used

### Internal I/O Driver ARP Send

**Icon**

**Description**
This event represents an internal NetX I/O driver ARP send event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Pointer to packet
- Info Field 3: Packet size
- Info Field 4: Not used

### Internal I/O Driver ARP Response Send

**Icon**

**Description**
This event represents an internal NetX I/O driver ARP response send event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Pointer to packet
- Info Field 3: Packet size
- Info Field 4: Not used
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal I/O Driver RARP Send</td>
<td>This event represents an internal NetX I/O driver RARP send event.</td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Pointer to packet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Packet size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>Internal I/O Driver Multicast Join</td>
<td>This event represents an internal NetX I/O driver multicast join event.</td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>Internal I/O Driver Multicast Leave</td>
<td>This event represents an internal NetX I/O driver multicast leave event.</td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>Internal I/O Driver Get Status</td>
<td>This event represents an internal NetX I/O driver get status event.</td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>Internal I/O Driver Get Speed</td>
<td>This event represents an internal NetX I/O driver get speed event.</td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>Internal I/O Driver Get Duplex Type</td>
<td>This event represents an internal NetX I/O driver get duplex type event.</td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
</tbody>
</table>
### Internal I/O Driver Get Error Count

**Icon:**

**Description:**
This event represents an internal NetX I/O driver get error count event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

---

### Internal I/O Driver Get RX Count

**Icon:**

**Description:**
This event represents an internal NetX I/O driver get RX count event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

---

### Internal I/O Driver Get TX Count

**Icon:**

**Description:**
This event represents an internal NetX I/O driver get TX count event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

---

### Internal I/O Driver Get Allocation Errors

**Icon:**

**Description:**
This event represents an internal NetX I/O driver get allocation errors event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

---

### Internal I/O Driver Un-initialize

**Icon:**

**Description:**
This event represents an internal NetX I/O driver un-initialize event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

---

### Internal I/O Driver Deferred Processing

**Icon:**

**Description:**
This event represents an internal NetX I/O driver deferred processing event.

**Information Fields**
- Info Field 1: Pointer to the IP instance
- Info Field 2: Pointer to packet
- Info Field 3: Packet length
- Info Field 4: Not used
### ARP Dynamic Entries Invalidate

**Icon**: 

**nx_arp_dynamic_entries_invalidate**

**Description**

This event represents invalidating all dynamic ARP entries via nx_arp_dynamic_entries_invalidate.

**Information Fields**

- Info Field 1: Pointer to the IP instance
- Info Field 2: Entries invalidated
- Info Field 3: Not used
- Info Field 4: Not used

### ARP Dynamic Entry Set

**Icon**: 

**nx_arp_dynamic_entry_set**

**Description**

This event represents setting a dynamic ARP entry via nx_arp_dynamic_entry_set.

**Information Fields**

- Info Field 1: Pointer to the IP instance
- Info Field 2: IP address
- Info Field 3: Physical address (MSW)
- Info Field 4: Physical address (LSW)

### ARP Enable

**Icon**: 

**nx_arp_enable**

**Description**

This event represents enabling ARP via nx_arp_enable.

**Information Fields**

- Info Field 1: Pointer to the IP instance
- Info Field 2: ARP cache memory pointer
- Info Field 3: ARP cache memory size
- Info Field 4: Not used

### ARP Gratuitous Send

**Icon**: 

**nx_arp_gratuitous_send**

**Description**

This event represents a gratuitous ARP send via nx_arp_gratuitous_send.

**Information Fields**

- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

### ARP Hardware Address Find

**Icon**: 

**nx_arp_hardware_address_find**

**Description**

This event represents finding a physical address via nx_arp_hardware_address_find.

**Information Fields**

- Info Field 1: Pointer to the IP instance
- Info Field 2: IP address
- Info Field 3: Physical address (MSW)
- Info Field 4: Physical address (LSW)

### ARP Information Get

**Icon**: 

**nx_arp_info_get**

**Description**

This event represents getting information via nx_arp_info_get.

**Information Fields**

- Info Field 1: Pointer to the IP instance
- Info Field 2: ARPs sent
- Info Field 3: ARP responses
- Info Field 4: ARPs received
### ARP IP Address Find

**Icon**

**nx_arp_ip_address_find**

**Description**

This event represents finding an IP address associated with the supplied physical address via *nx_arp_ip_address_find*.

**Information Fields**

- Info Field 1: Pointer to the IP instance
- Info Field 2: IP address
- Info Field 3: Physical address (MSW)
- Info Field 4: Physical address (LSW)

### ARP Static Entries Delete

**Icon**

**nx_arp_static_entries_delete**

**Description**

This event represents deleting all ARP static entries via *nx_arp_static_entries_delete*.

**Information Fields**

- Info Field 1: Pointer to the IP instance
- Info Field 2: Entries deleted
- Info Field 3: Not used
- Info Field 4: Not used

### ARP Static Entry Create

**Icon**

**nx_arp_static_entry_create**

**Description**

This event represents creating a static ARP entry via *nx_arp_static_entry_create*.

**Information Fields**

- Info Field 1: Pointer to the IP instance
- Info Field 2: IP address
- Info Field 3: Physical address (MSW)
- Info Field 4: Physical address (LSW)

### ARP Static Entry Delete

**Icon**

**nx_arp_static_entry_delete**

**Description**

This event represents deleting a static ARP entry via *nx_arp_static_entry_delete*.

**Information Fields**

- Info Field 1: Pointer to the IP instance
- Info Field 2: IP address
- Info Field 3: Physical address (MSW)
- Info Field 4: Physical address (LSW)

### Duo Cache Entry Delete

**Icon**

**nxd_nd_cache_entry_delete**

**Description**

This event represents deleting an entry in the neighbor cache table via *nxd_nd_cache_entry_create*.

**Information Fields**

- Info Field 1: Fourth (least significant) word of the IPv6 link local address to delete
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

### Duo Cache Entry Set

**Icon**

**nxd_nd_cache_entry_set**

**Description**

This event represents creating a cache entry and adding to the neighbor cache table via *nxd_nd_cache_entry_set*.

**Information Fields**

- Info Field 1: Fourth (least significant) word of the IPv6 address to add
- Info Field 2: Physical address msb
- Info Field 3: Physical address lsb
- Info Field 4: Not used
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Duo Cache Invalidate    | ![Icon](image1.png) | This event represents invalidating the entire neighbor cache table via nxd_nd_cache_invalidate. | Info Field 1: Pointer to IP instance  
Info Field 2: Not used  
Info Field 3: Not used  
Info Field 4: Not used |
| Duo Cache IP Address Find | ![Icon](image2.png) | This event represents retrieving an IP address matching the supplied physical address from the cache table via nxd_nd_cache_ip_address_find. | Info Field 1: Pointer to IP instance  
Info Field 2: Fourth (least significant) word of the IPv6 address  
Info Field 3: Physical address msb  
Info Field 4: Physical address lsb |
| Duo ICMP Enable         | ![Icon](image3.png) | This event represents ICMPv4 and ICMPv6 services being enabled on the specified IP instance via nxd_icmp_enable. | Info Field 1: Pointer to IP instance  
Info Field 2: Not used  
Info Field 3: Not used  
Info Field 4: Not used |
| Duo ICMP Ping           | ![Icon](image4.png) | This event represents sending a ping (echo request) to an IPv6 host via nxd_icmp_ping. | Info Field 1: Pointer to IP instance  
Info Field 2: IPv6 address  
Info Field 3: Pointer to echo data  
Info Field 4: Size of echo data |
| Duo IP Max Payload Size Find | ![Icon](image5.png) | This event computes the max payload the specified packet can carry without requiring fragmentation. | Info Field 1: Socket pointer  
Info Field 2: Peer IP address  
Info Field 3: Peer port  
Info Field 4: Not used |
| Duo IP Raw Packet Send  | ![Icon](image6.png) | This event represents sending a raw IP packet out the specified network interface to the supplied IP destination address via nxd_ip_raw_packet_send. | Info Field 1: Pointer to IP instance  
Info Field 2: Pointer to packet to send  
Info Field 3: Pointer to destination address  
Info Field 4: Packet protocol |
<table>
<thead>
<tr>
<th>Event Description</th>
<th>Icon</th>
<th>Icon Description</th>
<th>Icon</th>
<th>Icon Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Duo IPv6 Default Router Delete</strong></td>
<td><img src="image" alt="nxd_ipv6_default_router_delete" /></td>
<td><strong>nxd_ipv6_default_router_delete</strong>&lt;br&gt;&lt;br&gt;<strong>Description</strong>&lt;br&gt;This event represents removing a default router from the IP instance’s IPv6 routing table via <code>nxd_ipv6_default_router_delete</code>.</td>
<td><img src="image" alt="nxd_ipv6_enable" /></td>
<td><strong>nxd_ipv6_enable</strong>&lt;br&gt;&lt;br&gt;<strong>Description</strong>&lt;br&gt;This event represents enabling IPv6 services on the supplied IP instance via <code>nxd_ipv6_enable</code>.</td>
</tr>
<tr>
<td><strong>Duo IPv6 Global Address Get</strong></td>
<td><img src="image" alt="nxd_ipv6_global_address_get" /></td>
<td><strong>nxd_ipv6_global_address_get</strong>&lt;br&gt;&lt;br&gt;<strong>Description</strong>&lt;br&gt;This event represents retrieving the global (primary) IP address on the IP instance located at index 1 in the IP instance interface table via <code>nxd_ipv6_global_address_get</code>.</td>
<td><img src="image" alt="nxd_ipv6_global_address_set" /></td>
<td><strong>nxd_ipv6_global_address_set</strong>&lt;br&gt;&lt;br&gt;<strong>Description</strong>&lt;br&gt;This event represents setting the global (primary) IP address on the IP instance located at index 1 in the IP instance interface table via <code>nxd_ipv6_global_address_set</code>.</td>
</tr>
<tr>
<td><strong>Duo IPv6 Initiate Dad Process</strong></td>
<td><img src="image" alt="nxd_ipv6_initiate_dad_process" /></td>
<td><strong>nxd_ipv6_initiate_dad_process</strong>&lt;br&gt;&lt;br&gt;<strong>Description</strong>&lt;br&gt;This event represents the start of the Duplicate Address Detection (DAD) process when the IP instance is assigned a link local or an IP interface address via <code>nxd_ipv6_initiate_dad_process</code>.</td>
<td><img src="image" alt="nxd_ipv6_interface_address_get" /></td>
<td><strong>nxd_ipv6_interface_address_get</strong>&lt;br&gt;&lt;br&gt;<strong>Description</strong>&lt;br&gt;This event represents retrieving the IP address and prefix at the specified index into the IP instance interface address table via <code>nxd_ipv6_interface_address_get</code>.</td>
</tr>
</tbody>
</table>

**Information Fields**<br><br>**Duo IPv6 Default Router Delete**<br>Info Field 1: Pointer to IP instance.<br>Info Field 2: Fourth word (least significant) of the default router IPv6 address.<br>Info Field 3: Not used.<br>Info Field 4: Not used.<br><br>**Duo IPv6 Enable**<br>Info Field 1: Pointer to IP instance.<br>Info Field 2: Not used.<br>Info Field 3: Not used.<br>Info Field 4: Not used.<br><br>**Duo IPv6 Global Address Get**<br>Info Field 1: Pointer to IP instance.<br>Info Field 2: Fourth word (least significant) of the global address.<br>Info Field 3: IPv6 address prefix length.<br>Info Field 4: Index into IP interface table (1).<br><br>**Duo IPv6 Global Address Set**<br>Info Field 1: Pointer to IP instance.<br>Info Field 2: Not used.<br>Info Field 3: IPv6 address prefix length.<br>Info Field 4: Index into IP interface table (1).<br><br>**Duo IPv6 Initiate Dad Process**<br>Info Field 1: Pointer to IP instance.<br>Info Field 2: Not used.<br>Info Field 3: Not used.<br>Info Field 4: Not used.<br><br>**Duo IPv6 Interface Address Get**<br>Info Field 1: Pointer to IP instance.<br>Info Field 2: Fourth word (least significant) of the IPv6 address to return<br>Info Field 3: Prefix length.<br>Info Field 4: Index of interface into the IP instance interface table.
### Duo IPv6 Interface Address Set

**Icon**

**Description**
This event represents setting the IP address and prefix at the specified index into the IP instance interface address table. Not permitted on index zero (link local address) via `nxd_ipv6_interface_address_set`.

**Information Fields**
- Info Field 1: Pointer to IP instance
- Info Field 2: Fourth word (least significant) of the IPv6 address to return
- Info Field 3: Prefix length
- Info Field 4: Index of interface into the IP instance interface table

### Duo IPv6 Link Local Address Get

**Icon**

**Description**
This event represents retrieving the link local address of the specified IP instance via `nxd_ipv6_linklocal_address_get`.

**Information Fields**
- Info Field 1: Pointer to IP instance
- Info Field 2: Fourth word (least significant) of the IPv6 link local address
- Info Field 3: Not used
- Info Field 4: Not used

### Duo IPv6 Link Local Address Set

**Icon**

**Description**
This event represents setting the link local address of the IP instance via `nxd_ipv6_linklocal_address_set`.

**Information Fields**
- Info Field 1: Pointer to IP instance
- Info Field 2: Fourth (least significant) word of the IPv6 link local address
- Info Field 3: Not used
- Info Field 4: Not used

### Duo TCP Socket Peer Info Get

**Icon**

**Description**
This event extracts the sender data from a received TCP packet on the specified socket. It returns the IP address and port of the sender.

**Information Fields**
- Info Field 1: Socket pointer
- Info Field 2: Peer IP address
- Info Field 3: Peer port
- Info Field 4: Not used

### Duo TCP Socket Set Interface

**Icon**

**Description**
This event represents setting the outgoing socket interface after a client connects with a TCP server on the specified server IP address via `nxd_tcp_client_socket_connect`.

**Information Fields**
- Info Field 1: Pointer to TCP Socket
- Info Field 2: Interface ID
- Info Field 3: Not used
- Info Field 4: Not used

### Duo UDP Socket Set Interface

**Icon**

**Description**
This event represents setting the specified UDP socket outgoing interface to the interface corresponding to the input interface ID via `nxd_udp_socket_set_interface`.

**Information Fields**
- Info Field 1: Pointer to UDP Socket
- Info Field 2: Interface ID
- Info Field 3: Not used
- Info Field 4: Not used
<table>
<thead>
<tr>
<th>Event</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
<th>Icon</th>
</tr>
</thead>
</table>
| Duo UDP Source Extract                     | ![Icon](image) | This event represents extracting the IP address and source port of a received packet (either IPv4 or IPv6). If IPv6, the fourth word (least significant) of the IP address is returned via nxd_udp_source_extract. | Info Field 1: Pointer to the packet  
Info Field 2: IP version  
Info Field 3: Source IP address (IPv4 or IPv6)  
Info Field 4: Source port | ![Icon](image) |
| Duo UDP Source Send                        | ![Icon](image) | This event represents enabling checksum processing on a socket via nxd_udp_source_send.                                                                 | Info Field 1: Pointer to IP socket  
Info Field 2: Pointer to packet  
Info Field 3: Packet Size  
Info Field 4: IP Address | ![Icon](image) |
| ICMP Enable                                | ![Icon](image) | This event represents enabling ICMP via nx_icmp_enable.                                                                                                                        | Info Field 1: Pointer to the IP instance  
Info Field 2: Not used  
Info Field 3: Not used  
Info Field 4: Not used | ![Icon](image) |
| ICMP Information Get                       | ![Icon](image) | This event represents getting information via nx_icmp_info_get.                                                                                                               | Info Field 1: Pointer to the IP instance  
Info Field 2: Pings sent  
Info Field 3: Ping responses  
Info Field 4: Pings received | ![Icon](image) |
| ICMP Ping                                  | ![Icon](image) | This event represents pinging a target IP address via nx_icmp_ping.                                                                                                           | Info Field 1: Pointer to the IP instance  
Info Field 2: IP address  
Info Field 3: Pointer to data  
Info Field 4: Size of data | ![Icon](image) |
| IGMP Enable                                | ![Icon](image) | This event represents enabling IGMP via nxигм_enable.                                                                                                                              | Info Field 1: Pointer to the IP instance  
Info Field 2: Not used  
Info Field 3: Not used  
Info Field 4: Not used | ![Icon](image) |
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Function Name</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGMP Information Get</td>
<td>nx_igmp_info_get</td>
<td><img src="icon1.png" alt="Icon" /></td>
<td>This event represents getting information via nx_igmp_info_get.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td></td>
<td></td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Reports sent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Queries received</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Groups joined</td>
</tr>
<tr>
<td>IGMP Loopback Disable</td>
<td>nx_igmp_loopback_disable</td>
<td><img src="icon2.png" alt="Icon" /></td>
<td>This event represents disabling IGMP loopback via nx_igmp_loopback_disable.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td></td>
<td></td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>IGMP Loopback Enable</td>
<td>nx_igmp_loopback_enable</td>
<td><img src="icon3.png" alt="Icon" /></td>
<td>This event represents enabling IGMP loopback via nx_igmp_loopback_enable.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td></td>
<td></td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>IGMP Multicast Join</td>
<td>nx_igmp_multicast_join</td>
<td><img src="icon4.png" alt="Icon" /></td>
<td>This event represents joining a multicast group via nx_igmp_multicast_join.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td></td>
<td></td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Group IP address</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>IGMP Multicast Leave</td>
<td>nx_igmp_multicast_leave</td>
<td><img src="icon5.png" alt="Icon" /></td>
<td>This event represents leaving a multicast group via nx_igmp_multicast_leave.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td></td>
<td></td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Group IP address</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>IP Address Change Notify</td>
<td>nx_ip_address_change_notify</td>
<td><img src="icon6.png" alt="Icon" /></td>
<td>This event represents registering for IP change notification via nx_ip_address_change_notify.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td></td>
<td></td>
<td>Info Field 1: Pointer to the IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Callback function pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Additional information pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>Event Type</td>
<td>Function Call</td>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------</td>
<td>------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| IP Address Get       | `nx_ip_address_get`    | A    | This event represents getting the IP address via `nx_ip_address_get`.        | Info Field 1: Pointer to the IP instance  
                          |                        |      |                                                                             | Info Field 2: IP address  
                          |                        |      |                                                                             | Info Field 3: Network mask  
                          |                        |      |                                                                             | Info Field 4: Not used |
| IP Address Set       | `nx_ip_address_set`    | A    | This event represents setting the IP address via `nx_ip_address_set`.        | Info Field 1: Pointer to the IP instance  
                          |                        |      |                                                                             | Info Field 2: IP address  
                          |                        |      |                                                                             | Info Field 3: Network mask  
                          |                        |      |                                                                             | Info Field 4: Not used |
| IP Create            | `nx_ip_create`         | E    | This event represents creating an IP instance via `nx_ip_create`.             | Info Field 1: Pointer to the IP instance  
                          |                        |      |                                                                             | Info Field 2: IP address  
                          |                        |      |                                                                             | Info Field 3: Network mask  
                          |                        |      |                                                                             | Info Field 4: Default packet pool pointer |
| IP Delete            | `nx_ip_delete`         | E    | This event represents deleting an IP instance via `nx_ip_delete`.             | Info Field 1: Pointer to the IP instance  
                          |                        |      |                                                                             | Info Field 2: Not used  
                          |                        |      |                                                                             | Info Field 3: Not used  
                          |                        |      |                                                                             | Info Field 4: Not used |
| IP Driver Direct Command | `nx_ip_driver_direct_command` | D    | This event represents a direct I/O driver command via `nx_ip_driver_direct_command`. | Info Field 1: Pointer to the IP instance  
                          |                        |      |                                                                             | Info Field 2: Driver command  
                          |                        |      |                                                                             | Info Field 3: Return value  
                          |                        |      |                                                                             | Info Field 4: Not used |
| IP Forwarding Disable | `nx_ip_forwarding_disable` | F    | This event represents disabling IP forwarding via `nx_ip_forwarding_disable`. | Info Field 1: Pointer to the IP instance  
                          |                        |      |                                                                             | Info Field 2: Not used  
                          |                        |      |                                                                             | Info Field 3: Not used  
<pre><code>                      |                        |      |                                                                             | Info Field 4: Not used |
</code></pre>
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Function Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| IP Forwarding Enable       | nx_ip_address_get              |      | This event represents enabling IP forwarding via nx_ip_forwarding_enable.    | Info Field 1: Pointer to the IP instance  
Info Field 2: Not used  
Info Field 3: Not used  
Info Field 4: Not used |
| IP Fragment Disable        | nx_ip_fragment_disable         |      | This event represents disabling IP fragmenting via nx_ip_fragment_disable.   | Info Field 1: Pointer to the IP instance  
Info Field 2: Not used  
Info Field 3: Not used  
Info Field 4: Not used |
| IP Fragment Enable         | nx_ip_fragment_enable          |      | This event represents enabling IP fragmenting via nx_ip_fragment_enable.     | Info Field 1: Pointer to the IP instance  
Info Field 2: Not used  
Info Field 3: Not used  
Info Field 4: Not used |
| IP Gateway Address Set     | nx_ip_gateway_address_set      |      | This event represents setting the gateway IP address via nx_ip_gateway_address_set. | Info Field 1: Pointer to the IP instance  
Info Field 2: Gateway IP address  
Info Field 3: Not used  
Info Field 4: Not used |
| IP Information Get         | nx_ip_info_get                 |      | This event represents getting IP information via nx_ip_info_get.             | Info Field 1: Pointer to the IP instance  
Info Field 2: IP bytes sent  
Info Field 3: IP bytes received  
Info Field 4: IP packets dropped |
| IP Interface Attach        | nx_ip_interface_attach         |      | This event represents a secondary network interface being attached to the IP instance via nx_ip_interface_attach. | Info Field 1: Pointer to IP instance  
Info Field 2: Interface IP Address  
Info Field 3: Index into IP interface table  
Info Field 4: Not used |
### IP Interface Info Get

**Icon**: 🗒️

**Description**: This event represents information retrieved from the specified network interface via `nx_ip_interface_info_get`.

**Information Fields**:
- Info Field 1: Pointer to IP instance
- Info Field 2: Interface IP address
- Info Field 3: Interface MAC address msb
- Info Field 4: Interface MAC address lsb

---

### IP Raw Packet Disable

**Icon**: 🗑️

**Description**: This event represents disabling raw IP packet communication via `nx_ip_raw_packet_disable`.

**Information Fields**:
- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

---

### IP Raw Packet Enable

**Icon**: 🎨

**Description**: This event represents enabling raw IP packet communication via `nx_ip_raw_packet_enable`.

**Information Fields**:
- Info Field 1: Pointer to the IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

---

### IP Raw Packet Receive

**Icon**: 📫

**Description**: This event represents receiving a raw IP packet via `nx_ip_raw_packet_receive`.

**Information Fields**:
- Info Field 1: Pointer to the IP instance
- Info Field 2: Pointer to packet
- Info Field 3: Wait option
- Info Field 4: Not used

---

### IP Raw Packet Send

**Icon**: 🚀

**Description**: This event represents sending a raw IP packet via `nx_ip_raw_packet_send`.

**Information Fields**:
- Info Field 1: Pointer to the IP instance
- Info Field 2: Pointer to packet
- Info Field 3: Destination IP address
- Info Field 4: Type of service

---

### IP Static Route Add

**Icon**: 📢

**Description**: This event represents a static route being added to the IP instance routing table via `nx_ip_static_route_add`.

**Information Fields**:
- Info Field 1: Pointer to IP instance
- Info Field 2: Network address
- Info Field 3: Network mask
- Info Field 4: Next hop
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Function</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| IP Static Route Delete              | nx_ip_static_route_delete                    | This event represents a static route being removed from the IP instance routing table via nx_ip_static_route_delete. | Info Field 1: Pointer to IP instance  
Info Field 2: Network address  
Info Field 3: Network mask  
Info Field 4: Not used |
| IP Status Check                     | nx_ip_status_check                           | This event represents checking for an IP status via nx_ip_status_check.                                  | Info Field 1: Pointer to the IP instance  
Info Field 2: Requested status  
Info Field 3: Actual status  
Info Field 4: Wait option |
| IPSEC Enable                        | nx_ipsec_enable                              | This event represents enabling IPSec services on the supplied IP instance via nx_ipsec_enable.           | Info Field 1: Pointer to IP instance  
Info Field 2: Not used  
Info Field 3: Not used  
Info Field 4: Not used |
| Packet Allocate                     | nx_packet_allocate                           | This event represents allocating a packet via nx_packet_allocate.                                      | Info Field 1: Pointer to the packet pool  
Info Field 2: Pointer to packet allocated  
Info Field 3: Packet type  
Info Field 4: Available packets |
| Packet Copy                         | nx_packet_copy                               | This event represents copying a packet via nx_packet_copy.                                             | Info Field 2: New packet pointer  
Info Field 3: Pointer to packet pool  
Info Field 4: Wait option |
| Packet Data Append                  | nx_packet_data_append                        | This event represents appending data to a packet via nx_packet_data_append.                            | Info Field 1: Pointer to the packet  
Info Field 2: Pointer to data  
Info Field 3: Size of data  
Info Field 4: Pointer to packet pool |
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Packet Data Extract Offset   | `nx_udp_source_extract_offset` | ![Icon](Image) | This event represents packet data that is extracted into a supplied buffer from a packet via `nx_udp_source_extract_offset`. | Info Field 1: Pointer to packet  
Info Field 2: Size of specified buffer  
Info Field 3: Number of bytes copied  
Info Field 4: Not used |
| Packet Data Retrieve         | `nx_packet_data_retrieve`    | ![Icon](Image) | This event represents retrieving data from a packet via `nx_packet_data_retrieve`. | Info Field 1: Pointer to the packet  
Info Field 2: Pointer to start of buffer  
Info Field 3: Bytes copied  
Info Field 4: Not used |
| Packet Length Get            | `nx_packet_length_get`       | ![Icon](Image) | This event represents getting the length of a packet via `nx_packet_length_get`. | Info Field 1: Pointer to the packet  
Info Field 2: Packet length  
Info Field 3: Not used  
Info Field 4: Not used |
| Packet Pool Create           | `nx_packet_pool_create`      | ![Icon](Image) | This event represents creating a packet pool via `nx_packet_pool_create`. | Info Field 1: Pointer to the packet pool  
Info Field 2: Packet payload size  
Info Field 3: Pointer to pool memory area  
Info Field 4: Size of pool memory area |
| Packet Pool Delete           | `nx_packet_pool_delete`      | ![Icon](Image) | This event represents deleting a packet pool via `nx_packet_pool_delete`. | Info Field 1: Pointer to the packet pool  
Info Field 2: Not used  
Info Field 3: Not used  
Info Field 4: Not used |
| Packet Pool Information Get  | `nx_packet_pool_info_get`    | ![Icon](Image) | This event represents getting packet pool information via `nx_packet_pool_info_get`. | Info Field 1: Pointer to packet pool  
Info Field 2: Total packets  
Info Field 3: Available packets  
Info Field 4: Empty requests |
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet Release</td>
<td>nx_packet_release</td>
<td></td>
<td>This event represents releasing a packet via nx_packet_release.</td>
<td>Info Field 1: Pointer to the packet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Packet status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Available packets</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>Packet Transmit Release</td>
<td>nx_packet_transmit_release</td>
<td></td>
<td>This event represents releasing a transmit packet via nx_packet_transmit_release.</td>
<td>Info Field 1: Pointer to the packet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Packet status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Available packets</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>RARP Disable</td>
<td>nx_rarp_disable</td>
<td></td>
<td>This event represents disabling RARP via nx_rarp_disable.</td>
<td>Info Field 1: Pointer to IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>RARP Enable</td>
<td>nx_rarp_enable</td>
<td></td>
<td>This event represents enabling RARP via nx_rarp_enable.</td>
<td>Info Field 1: Pointer to IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
<tr>
<td>RARP Information Get</td>
<td>nx_rarp_info_get</td>
<td></td>
<td>This event represents getting RARP information via nx_rarp_info_get.</td>
<td>Info Field 1: Pointer to IP instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Requests sent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Responses received</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Invalid responses</td>
</tr>
<tr>
<td>System Initialize</td>
<td>nx_system_initialize</td>
<td></td>
<td>This event represents initializing NetX via nx_system_initialize.</td>
<td>Info Field 1: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used</td>
</tr>
</tbody>
</table>
### TCP Client Socket Bind

**Icon**

**nx_tcp_client_socket_bind**

**Description**

This event represents binding a client socket to a port via `nx_tcp_client_socket_bind`.

**Information Fields**

- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Port requested
- Info Field 4: Wait option

### TCP Client Socket Connect

**Icon**

**nx_tcp_client_socket_connect**

**Description**

This event represents making a client socket connection via `nx_tcp_client_socket_connect`.

**Information Fields**

- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Server IP address
- Info Field 4: Server port requested

### TCP Client Socket Port Get

**Icon**

**nx_tcp_client_socket_port_get**

**Description**

This event represents getting the client socket port number via `nx_tcp_client_socket_port_get`.

**Information Fields**

- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Port number
- Info Field 4: Not used

### TCP Client Socket Unbind

**Icon**

**nx_tcp_client_socket_unbind**

**Description**

This event represents unbinding the port associated with the socket via `nx_tcp_client_socket_unbind`.

**Information Fields**

- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Not used
- Info Field 4: Not used

### TCP Enable

**Icon**

**nx_tcp_enable**

**Description**

This event represents enabling TCP via `nx_tcp_enable`.

**Information Fields**

- Info Field 1: Pointer to IP instance
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

### TCP Free Port Find

**Icon**

**nx_tcp_free_port_find**

**Description**

This event represents finding a free TCP port via `nx_tcp_free_port_find`.

**Information Fields**

- Info Field 1: Pointer to IP instance
- Info Field 2: Starting search port number
- Info Field 3: Free port number
- Info Field 4: Not used
### TCP Socket Bytes Available

- **Icon**: ![Icon](nx_tcp_socket_bytes_available)
- **Description**: This event represents the number of bytes currently available on the specified TCP receiving socket via `nx_tcp_socket_bytes_available`.
- **Information Fields**:
  - Info Field 1: Pointer to IP instance
  - Info Field 2: Pointer to TCP socket
  - Info Field 3: Bytes received on the socket
  - Info Field 4: Not used

### TCP Socket Create

- **Icon**: ![Icon](nx_tcp_socket_create)
- **Description**: This event represents creating a TCP socket via `nx_tcp_socket_create`.
- **Information Fields**:
  - Info Field 1: Pointer to IP instance
  - Info Field 2: Pointer to socket
  - Info Field 3: Type of service
  - Info Field 4: Receive window size

### TCP Socket Delete

- **Icon**: ![Icon](nx_tcp_socket_delete)
- **Description**: This event represents deleting a socket via `nx_tcp_socket_delete`.
- **Information Fields**:
  - Info Field 1: Pointer to IP instance
  - Info Field 2: Pointer to socket
  - Info Field 3: Socket state
  - Info Field 4: Not used

### TCP Socket Disconnect

- **Icon**: ![Icon](nx_tcp_socket_disconnect)
- **Description**: This event represents disconnecting a socket via `nx_tcp_socket_disconnect`.
- **Information Fields**:
  - Info Field 1: Pointer to IP instance
  - Info Field 2: Pointer to socket
  - Info Field 3: Wait option
  - Info Field 4: Socket state

### TCP Socket Information Get

- **Icon**: ![Icon](nx_tcp_socket_info_get)
- **Description**: This event represents getting information about a socket via `nx_tcp_socket_info_get`.
- **Information Fields**:
  - Info Field 1: Pointer to IP instance
  - Info Field 2: Pointer to socket
  - Info Field 3: Bytes sent through this socket
  - Info Field 4: Bytes received through this socket

### TCP Socket MSS Get

- **Icon**: ![Icon](nx_tcp_socket_mss_get)
- **Description**: This event represents getting the socket's MSS via `nx_tcp_socket_mss_get`.
- **Information Fields**:
  - Info Field 1: Pointer to IP instance
  - Info Field 2: Pointer to socket
  - Info Field 3: Maximum Segment Size (MSS)
  - Info Field 4: Socket state
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| TCP Socket MSS Peer Get           | This event represents getting the MSS value of the socket's peer via nx_tcp_socket_mss_peer_get. | Info Field 1: Pointer to IP instance  
Info Field 2: Pointer to socket  
Info Field 3: Peer's MSS  
Info Field 4: Socket state |
| TCP Socket MSS Set                | This event represents setting a socket's MSS via nx_tcp_socket_mss_set.       | Info Field 1: Pointer to IP instance  
Info Field 2: Pointer to socket  
Info Field 3: MSS  
Info Field 4: Socket state |
| TCP Socket Peer Info Get          | This event represents information retrieved from the TCP socket regarding the peer (e.g. connecting host) IP address and port via nx_tcp_socket_peer_info_get. | Info Field 1: Pointer to TCP socket  
Info Field 2: Peer IP address  
Info Field 3: Peer port number  
Info Field 4: Not used |
| TCP Socket Receive Notify         | This event represents registering a receive notify callback for a socket via nx_tcp_socket_receive_notify. | Info Field 1: Pointer to IP instance  
Info Field 2: Pointer to socket  
Info Field 3: Pointer to receive notify callback  
Info Field 4: Not used |
| TCP Socket Receive                | This event represents receiving data from a socket via nx_tcp_socket_receive. | Info Field 1: Pointer to socket  
Info Field 2: Pointer to received packet  
Info Field 3: Received packet length  
Info Field 4: Receive sequence number |
| TCP Socket Send                   | This event represents sending data on a socket via nx_tcp_socket_send.        | Info Field 1: Pointer to socket  
Info Field 2: Pointer to packet  
Info Field 3: Length of packet  
Info Field 4: Transmit sequence number |
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| TCP Socket State Wait              |      | This event represents waiting for a socket to enter a particular state via    | Info Field 1: Pointer to IP instance  
|                                    |      | nx_tcp_socket_state_wait                                                     | Info Field 2: Pointer to socket  
|                                    |      |                                                                             | Info Field 3: Desired socket state  
|                                    |      |                                                                             | Info Field 4: Previous socket state |
| TCP Socket Transmit Configure      |      | This event represents configuring the transmit options for a socket via       | Info Field 1: Pointer to IP instance  
|                                    |      | nx_tcp_socket_transmit_configure                                             | Info Field 2: Pointer to socket  
|                                    |      |                                                                             | Info Field 3: Transmit queue depth  
|                                    |      |                                                                             | Info Field 4: Timeout value        |
| TCP Socket Window Update Notify Set|      | This event represents a TCP socket receiving notification of an increase in  | Info Field 1: Pointer to TCP socket  
|                                    |      | the remote host receive window via nx_tcp_window_update_notify_set.           | Info Field 2: Not used  
|                                    |      |                                                                             | Info Field 3: Not used  
|                                    |      |                                                                             | Info Field 4: Not used  
| UDP Enable                         |      | This event represents enabling UDP via nx_udp_enable.                         | Info Field 1: Pointer to IP instance  
|                                    |      |                                                                             | Info Field 2: Not used  
|                                    |      |                                                                             | Info Field 3: Not used  
|                                    |      |                                                                             | Info Field 4: Not used  
| UDP Free Port Find                 |      | This event represents finding a free UDP port via nx_udp_free_port_find.     | Info Field 1: Pointer to IP instance  
|                                    |      |                                                                             | Info Field 2: Starting port to search from  
|                                    |      |                                                                             | Info Field 3: Free port  
|                                    |      |                                                                             | Info Field 4: Not used  
| UDP Information Get                |      | This event represents getting information via nx_udp_info_get.               | Info Field 1: Pointer to IP instance  
|                                    |      |                                                                             | Info Field 2: UDP bytes sent  
|                                    |      |                                                                             | Info Field 3: UDP bytes received  
|                                    |      |                                                                             | Info Field 4: Invalid packets    |
### UDP Socket Bind

**nx_udp_socket_bind**

**Description**
This event represents binding a UDP socket to a port via `nx_udp_socket_bind`.

**Information Fields**
- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Port number
- Info Field 4: Wait option

### UDP Socket Bytes Available

**nx_udp_socket_bytes_available**

**Description**
This event represents the current number of bytes received on the UDP socket via `nx_udp_socket_bytes_available`.

**Information Fields**
- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Bytes received on socket
- Info Field 4: Not used

### UDP Socket Checksum Disable

**nx_udp_socket_checksum_disable**

**Description**
This event represents disabling the checksum for data on a UDP socket via `nx_udp_socket_checksum_disable`.

**Information Fields**
- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Not used
- Info Field 4: Not used

### UDP Socket Checksum Enable

**nx_udp_socket_checksum_enable**

**Description**
This event represents enabling checksum processing on a socket via `nx_udp_socket_checksum_enable`.

**Information Fields**
- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Not used
- Info Field 4: Not used

### UDP Socket Create

**nx_udp_socket_create**

**Description**
This event represents creating a UDP socket via `nx_udp_socket_create`.

**Information Fields**
- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Type of service
- Info Field 4: Maximum receive queue

### UDP Socket Delete Event

**nx_udp_socket_delete event**

**Description**
This event represents deleting a UDP socket via `nx_udp_socket_delete`.

**Information Fields**
- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Not used
- Info Field 4: Not used
<table>
<thead>
<tr>
<th>Event Description</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP Socket Information Get Event</td>
<td>nx_udp_socket_info_get_event</td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This event represents getting information about a UDP socket via nx_udp_socket_info_get.</td>
</tr>
</tbody>
</table>
| **Information Fields** | Info Field 1: Pointer to IP instance  
Info Field 2: Pointer to socket  
Info Field 3: Bytes sent through socket  
Info Field 4: Bytes received through socket |

<table>
<thead>
<tr>
<th>Event Description</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP Socket Interface Set</td>
<td>nx_udp_socket_interface_set_event</td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This event represents setting the outgoing interface of the specified UDP socket with the specified interface via nx_udp_socket_interface_set.</td>
</tr>
</tbody>
</table>
| **Information Fields** | Info Field 1: Pointer to UDP socket  
Info Field 2: Index corresponding to the interface for the socket  
Info Field 3: Not used  
Info Field 4: Not used |

<table>
<thead>
<tr>
<th>Event Description</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP Socket Port Get</td>
<td>nx_udp_socket_port_get</td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This event represents retrieving the UDP port the specified UDP socket is bound to via nx_udp_socket_port_get.</td>
</tr>
</tbody>
</table>
| **Information Fields** | Info Field 1: Pointer to IP instance  
Info Field 2: Pointer to UDP socket  
Info Field 3: Port number  
Info Field 4: Not used |

<table>
<thead>
<tr>
<th>Event Description</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP Socket Receive</td>
<td>nx_udp_socket_receive</td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This event represents receiving data on the specified UDP socket via nx_udp_socket_receive.</td>
</tr>
</tbody>
</table>
| **Information Fields** | Info Field 1: Pointer to IP instance  
Info Field 2: Pointer to UDP socket  
Info Field 3: Pointer to received packet  
Info Field 4: Received packet size |

<table>
<thead>
<tr>
<th>Event Description</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP Socket Receive Notify</td>
<td>nx_udp_socket_receive_notify</td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This event represents registering a receive notify callback via nx_udp_socket_receive_notify.</td>
</tr>
</tbody>
</table>
| **Information Fields** | Info Field 1: Pointer to IP instance  
Info Field 2: Pointer to socket  
Info Field 3: Pointer to receive notify function  
Info Field 4: Not used |

<table>
<thead>
<tr>
<th>Event Description</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP Socket Send</td>
<td>nx_udp_socket_send</td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This event represents sending data through a UDP socket via nx_udp_socket_send.</td>
</tr>
</tbody>
</table>
| **Information Fields** | Info Field 1: Pointer to socket  
Info Field 2: Pointer to packet  
Info Field 3: Packet length  
Info Field 4: Destination IP address |
<table>
<thead>
<tr>
<th><strong>UDP Socket Unbind</strong></th>
<th><strong>UDP Source Extract</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Icon</td>
<td>nx_udp_socket_unbind</td>
</tr>
</tbody>
</table>
| **Description** | **Description**
| This event represents unbinding a UDP port with a socket via `nx_udp_socket_unbind`. | This event represents getting the IP address and port number of a received UDP packet via `nx_udp_source_extract`. |
| **Information Fields** | **Information Fields**
| Info Field 1: Pointer to IP instance | Info Field 1: Pointer to packet
| Info Field 2: Pointer to socket | Info Field 2: Sender's IP address
| Info Field 3: Port number | Info Field 3: Sender's port number
| Info Field 4: Not used | Info Field 4: Not used |
This chapter contains a description of the USBX events displayed by TraceX.

- List of Events and Icons 172
- Event Descriptions 182
List of Events and Icons

The following is a list of USBX events displayed by TraceX.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>Device Class Cdc Activate (ux_device_class_cdc_activate)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Cdc Deactivate (ux_device_class_cdc_deactivate)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Cdc Read (ux_device_class_cdc_read)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Cdc Write (ux_device_class_cdc_write)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Dpump Activate (ux_device_class_dpump_activate)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Dpump Deactivate (ux_device_class_dpump_deactivate)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Dpump Read (ux_device_class_dpump_read)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Dpump Write (ux_device_class_dpump_write)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Hid Activate (ux_device_class_hid_activate)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Hid Deactivate (ux_device_class_hid_deactivate)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Hid Descriptor Send (ux_device_class_hid_descriptor_send)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Hid Event Get (ux_device_class_hid_event_get)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Hid Event Set (ux_device_class_hid_event_set)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Hid Report Get (ux_device_class_hid_report_get)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Hid Report Set (ux_device_class_hid_report_set)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Pima Activate (ux_device_class_pima_activate)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Pima Deactivate (ux_device_class_pima_deactivate)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Pima Device Info Send (ux_device_class_pima_device_info_send)</td>
</tr>
<tr>
<td>🔄</td>
<td>Device Class Pima Event Get (ux_device_class_pima_event_get)</td>
</tr>
</tbody>
</table>
Device Class Storage Deactivate (ux_device_class_storage_deactivate)
Device Class Storage Format (ux_device_class_storage_format)
Device Class Storage Inquiry (ux_device_class_storage_inquiry)
Device Class Storage Mode Select (ux_device_class_storage_mode_select)
Device Class Storage Mode Sense (ux_device_class_storage_mode_sense)
Device Class Storage Prevent Allow Media Removal
(ux_device_class_storage_prevent_allow_media_removal)
Device Class Storage Read (ux_device_class_storage_read)
Device Class Storage Read Capacity (ux_device_class_storage_read_capacity)
Device Class Storage Read Format Capacity
(ux_device_class_storage_read_format_capacity)
Device Class Storage Read TOC (ux_device_class_storage_read_toc)
Device Class Storage Request Sense (ux_device_class_storage_request_sense)
Device Class Storage Start Stop (ux_device_class_storage_start_stop)
Device Class Storage Test Ready (ux_device_class_storage_test_ready)
Device Class Storage Verify (ux_device_class_storage_verify)
Device Class Storage Write (ux_device_class_storage_write)
Device Stack Alternate Setting Get (ux_device_stack_alternate_setting_get)
Device Stack Alternate Setting Set (ux_device_stack_alternate_setting_set)
Device Stack Class Register (ux_device_stack_class_register)
Device Stack Clear Feature (ux_device_stack_clear_feature)
Device Stack Configuration Get (ux_device_stack_configuration_get)
Device Stack Configuration Set (ux_device_stack_configuration_set)
Device Stack Connect (ux_device_stack_connect)
Device Stack Descriptor Send \((ux\_device\_stack\_descriptor\_send)\)

Device Stack Disconnect \((ux\_device\_stack\_disconnect)\)

Device Stack Endpoint Stall \((ux\_device\_stack\_endpoint\_stall)\)

Device Stack Get Status \((ux\_device\_stack\_get\_status)\)

Device Stack Host Wakeup \((ux\_device\_stack\_host\_wakeup)\)

Device Stack Initialize \((ux\_device\_stack\_initialize)\)

Device Stack Interface Delete \((ux\_device\_stack\_interface\_delete)\)

Device Stack Interface Get \((ux\_device\_stack\_interface\_get)\)

Device Stack Interface Set \((ux\_device\_stack\_interface\_set)\)

Device Stack Set Feature \((ux\_device\_stack\_set\_feature)\)

Device Stack Transfer Abort \((ux\_device\_stack\_transfer\_abort)\)

Device Stack Transfer All Request Abort \((ux\_device\_stack\_transfer\_all\_request\_abort)\)

Device Stack Transfer Request \((ux\_device\_stack\_transfer\_request)\)

Host Class Asix Activate \((ux\_host\_class\_asix\_activate)\)

Host Class Asix Deactivate \((ux\_host\_class\_asix\_deactivate)\)

Host Class Asix Interrupt Notification \((ux\_host\_class\_asix\_interrupt\_ notification)\)

Host Class Asix Read \((ux\_host\_class\_asix\_read)\)

Host Class Asix Write \((ux\_host\_class\_asix\_write)\)

Host Class Audio Activate \((ux\_host\_class\_audio\_activate)\)

Host Class Audio Control Value Get \((ux\_host\_class\_audio\_control\_value\_get)\)

Host Class Audio Control Value Set \((ux\_host\_class\_audio\_control\_value\_set)\)

Host Class Audio Deactivate \((ux\_host\_class\_audio\_deactivate)\)
Host Class Audio Read (ux_host_class_audio_read)
Host Class Audio Streaming Sampling Get (ux_host_class_audio_streaming_sampling_get)
Host Class Audio Streaming Sampling Set (ux_host_class_audio_streaming_sampling_set)
Host Class Audio Write (ux_host_class_audio_write)
Host Class Cdc Acm Activate (ux_host_class_cdc_acm_activate)
Host Class Cdc Acm Deactivate (ux_host_class_cdc_acm_deactivate)
Host Class Cdc Acm ioctl Abort In Pipe (ux_host_class_cdc_acm_ioctl_abort_in_pipe)
Host Class Cdc Acm ioctl Abort Out Pipe (ux_host_class_cdc_acm_ioctl_abort_out_pipe)
Host Class Cdc Acm ioctl Get Device Status (ux_host_class_cdc_acm_ioctl_get_device_status)
Host Class Cdc Acm ioctl Get Line Coding (ux_host_class_cdc_acm_ioctl_get_line_coding)
Host Class Cdc Acm ioctl Notification Callback (ux_host_class_cdc_acm_ioctl_notification_callback)
Host Class Cdc Acm ioctl Send Break (ux_host_class_cdc_acm_ioctl_send_break)
Host Class Cdc Acm ioctl Set Line Coding (ux_host_class_cdc_acm_ioctl_set_line_coding)
Host Class Cdc Acm ioctl Set Line State (ux_host_class_cdc_acm_ioctl_set_line_state)
Host Class Cdc Acm Read (ux_host_class_cdc_acm_read)
Host Class Cdc Acm Reception Start (ux_host_class_cdc_acm_reception_start)
Host Class Cdc Acm Reception Stop (ux_host_class_cdc_acm_reception_stop)
Host Class Cdc Acm Write (ux_host_class_cdc_acm_write)
Host Class Dpump Activate (ux_host_class_dpump_activate)
Host Class Dpump Deactivate (ux_host_class_dpump_deactivate)
Host Class Dpump Read (ux_host_class_dpump_read)
Host Class Dpump Write (ux_host_class_dpump_write)

Host Class Hid Activate (ux_host_class_hid_activate)

Host Class Hid Client Register (ux_host_class_hid_client_register)

Host Class Hid Deactivate (ux_host_class_hid_deactivate)

Host Class Hid Idle Get (ux_host_class_hid_idle_get)

Host Class Hid Idle Set (ux_host_class_hid_idle_set)

Host Class Hid Keyboard Activate (ux_host_class_hid_keyboard_activate)

Host Class Hid Keyboard Deactivate (ux_host_class_hid_keyboard_deactivate)

Host Class Hid Mouse Activate (ux_host_class_hid_mouse_activate)

Host Class Hid Mouse Deactivate (ux_host_class_hid_mouse_deactivate)

Host Class Hid Remote Control Activate (ux_host_class_hid_remote_control_activate)

Host Class Hid Remote Control Deactivate (ux_host_class_hid_remote_control_deactivate)

Host Class Hid Report Get (ux_host_class_hid_report_get)

Host Class Hid Report Set (ux_host_class_hid_report_set)

Host Class Hub Activate (ux_host_class_hub_activate)

Host Class Hub Change Detect (ux_host_class_hub_change_detect)

Host Class Hub Deactivate (ux_host_class_hub_deactivate)

Host Class Hub Port Change Connection Process (ux_host_class_hub_port_change_connection_process)

Host Class Hub Port Change Enable Process (ux_host_class_hub_port_change_enable_process)

Host Class Hub Port Change Over Current Process (ux_host_class_hub_port_change_over_current_process)

Host Class Hub Port Change Reset Process (ux_host_class_hub_port_change_reset_process)
Host Class Hub Port Change Suspend Process  
(ux_host_class_hub_port_change_suspend_process)

Host Class Pima Activate  (ux_host_class_pima_activate)

Host Class Pima Deactivate  (ux_host_class_pima_deactivate)

Host Class Pima Device Info Get  (ux_host_class_pima_device_info_get)

Host Class Pima Device Reset  (ux_host_class_pima_device_reset)

Host Class Pima Notification  (ux_host_class_pima_notification)

Host Class Pima Number Objects Get  (ux_host_class_pima_num_objects_get)

Host Class Pima Object Close  (ux_host_class_pima_object_close)

Host Class Pima Object Copy  (ux_host_class_pima_object_copy)

Host Class Pima Object Delete  (ux_host_class_pima_object_delete)

Host Class Pima Object Get  (ux_host_class_pima_object_get)

Host Class Pima Object Info Get  (ux_host_class_pima_object_info_get)

Host Class Pima Object Info Send  (ux_host_class_pima_object_info_send)

Host Class Pima Object Move  (ux_host_class_pima_object_move)

Host Class Pima Object Send  (ux_host_class_pima_object_send)

Host Class Pima Object Transfer Abort  (ux_host_class_object_transfer_abort)

Host Class Pima Read  (ux_host_class_pima_read)

Host Class Pima Request Cancel  (ux_host_class_pima_request_cancel)

Host Class Pima Session Close  (ux_host_class_pima_session_close)

Host Class Pima Session Open  (ux_host_class_pima_session_open)

Host Class Pima Storage Ids Get  (ux_host_class_pima_storage_ids_get)

Host Class Pima Storage Info Get  (ux_host_class_pima_storage_info_get)
Host Class Pima Thumb Get (ux_host_class_pima_thumb_get)
Host Class Pima Write (ux_host_class_pima_write)
Host Class Printer Activate (ux_host_class_printer_activate)
Host Class Printer Deactivate (ux_host_class_printer_deactivate)
Host Class Printer Name Get (ux_host_class_printer_name_get)
Host Class Printer Read (ux_host_class_printer_read)
Host Class Printer Soft Reset (ux_host_class_printer_soft_reset)
Host Class Printer Status Get (ux_host_class_printer_status_get)
Host Class Printer Write (ux_host_class_printer_write)
Host Class Prolific Activate (ux_host_class_prolific_activate)
Host Class Prolific Deactivate (ux_host_class_prolific_deactivate)
Host Class Prolific ioctl Abort In Pipe (ux_host_class_prolific_ioctl_abort_in_pipe)
Host Class Prolific ioctl Abort Out Pipe (ux_host_class_prolific_ioctl_abort_out_pipe)
Host Class Prolific ioctl Get Device Status (ux_host_class_prolific_ioctl_get_device_status)
Host Class Prolific ioctl Get Line Coding (ux_host_class_prolific_ioctl_get_line_coding)
Host Class Prolific ioctl Purge (ux_host_class_prolific_ioctl_purge)
Host Class Prolific ioctl Report Device Status Change (ux_host_class_prolific_ioctl_report_device_status_change)
Host Class Prolific ioctl Send Break (ux_host_class_prolific_ioctl_send_break)
Host Class Prolific ioctl Set Line Coding (ux_host_class_prolific_ioctl_set_line_coding)
Host Class Prolific ioctl Set Line State (ux_host_class_prolific_ioctl_set_line_state)
Host Class Prolific Read (ux_host_class_prolific_read)
Host Class Prolific Reception Start (ux_host_class_prolific_reception_start)
Host Class Prolific Reception Stop \((ux\_host\_class\_prolickr\_reception\_stop)\)

Host Class Prolific Write \((ux\_host\_class\_prolickr\_write)\)

Host Class Storage Activate \((ux\_host\_class\_storage\_activate)\)

Host Class Storage Deactivate \((ux\_host\_class\_storage\_deactivate)\)

Host Class Storage Media Capacity Get
\((ux\_host\_class\_storage\_media\_capacity\_get)\)

Host Class Storage Media Format Capacity Get
\((ux\_host\_class\_storage\_media\_format\_capacity\_get)\)

Host Class Storage Media Mount \((ux\_host\_class\_storage\_media\_mount)\)

Host Class Storage Media Open \((ux\_host\_class\_storage\_media\_open)\)

Host Class Storage Media Read \((ux\_host\_class\_storage\_media\_read)\)

Host Class Storage Media Write \((ux\_host\_class\_storage\_media\_write)\)

Host Class Storage Request Sense \((ux\_host\_class\_storage\_request\_sense)\)

Host Class Storage Start Stop \((ux\_host\_class\_storage\_start\_stop)\)

Host Class Storage Unit Ready Test \((ux\_host\_class\_storage\_activate)\)

Host Stack Class Instance Create \((ux\_host\_stack\_class\_instance\_create)\)

Host Stack Class Instance Destroy \((ux\_host\_stack\_class\_instance\_destroy)\)

Host Stack Configuration Delete \((ux\_host\_stack\_configuration\_delete)\)

Host Stack Configuration Enumerate \((ux\_host\_stack\_configuration\_enumerate)\)

Host Stack Configuration Instance Create
\((ux\_host\_stack\_configuration\_instance\_create)\)

Host Stack Configuration Instance Delete
\((ux\_host\_stack\_configuration\_instance\_delete)\)

Host Stack Configuration Set \((ux\_host\_stack\_configuration\_set)\)

Host Stack Device Address Set \((ux\_host\_stack\_device\_set)\)

Host Stack Device Configuration Get \((ux\_host\_stack\_device\_configuration\_get)\)

Host Stack Device Configuration Select
\((ux\_host\_stack\_device\_configuration\_select)\)
Host Stack Device Descriptor Read: (ux_host_stack_device_descriptor_read)

Host Stack Device Get: (ux_host_stack_device_get)

Host Stack Device Remove: (ux_host_stack_device_get)

Host Stack Device Resource Free: (ux_host_stack_device_resource_free)

Host Stack Endpoint Instance Create: (ux_host_stack_endpoint_instance_create)

Host Stack Endpoint Instance Delete: (ux_host_stack_endpoint_instance_delete)

Host Stack Endpoint Reset: (ux_host_stack_endpoint_reset)

Host Stack Endpoint Transfer Abort: (ux_host_stack_endpoint_transfer_abort)

Host Stack Host Controller Register: (ux_host_stack_hcd_register)

Host Stack Initialize: (ux_host_stack_initialize)

Host Stack Interface Endpoint Get: (ux_host_stack_interface_endpoint_get)

Host Stack Interface Instance Create: (ux_host_stack_interface_instance_create)

Host Stack Interface Instance Delete: (ux_host_stack_interface_instance_delete)

Host Stack Interface Set: (ux_host_stack_interface_set)

Host Stack Interface Setting Select: (ux_host_stack_interface_setting_select)

Host Stack New Configuration Create: (ux_host_stack_new_configuration_create)

Host Stack New Device Create: (ux_host_stack_new_device_create)

Host Stack New Endpoint Create: (ux_host_stack_new_endpoint_create)

Host Stack Root Hub Change Process: (ux_host_stack_rh_change_process)

Host Stack Root Hub Device Extraction: (ux_host_stack_rh_device_extraction)

Host Stack Root Hub Device Insertion: (ux_host_stack_rh_device_insertion)

Host Stack Transfer Request: (ux_host_stack_transfer_request)
Event Descriptions

The following pages describe the USBX Trace Events.
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Device Class Cdc Activate</strong></td>
<td>This event represents a USBX Device Class Cdc Activate Event.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Device Class Cdc Deactivate</strong></td>
<td>This event represents a USBX Device Class Cdc Deactivate.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Device Class Cdc Read</strong></td>
<td>This event represents a USBX Device Class Cdc Read Event.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Data pointer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Requested length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Device Class Cdc Write</strong></td>
<td>This event represents a USBX Device Class Cdc Write Event.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Data pointer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Requested length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Device Class Dpump Activate</strong></td>
<td>This event represents a USBX Device Class Dpump Activate Event.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Device Class Dpump Deactivate</strong></td>
<td>This event represents a USBX Device Class Dpump Deactivate Event.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Device Class Dpump Read</td>
<td>Device Class Dpump Write</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
<td></td>
</tr>
<tr>
<td>ux_device_class_dpump_read</td>
<td>ux_device_class_dpump_write</td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Dpump Read Event.</td>
<td>This event represents a USBX Device Class Dpump Write Event.</td>
<td></td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
<td></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
<td></td>
</tr>
<tr>
<td>Info Field 2: Buffer.</td>
<td>Info Field 2: Data pointer.</td>
<td></td>
</tr>
<tr>
<td>Info Field 3: Requested length.</td>
<td>Info Field 3: Requested length.</td>
<td></td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Class Hid Activate</th>
<th>Device Class Hid Deactivate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td>ux_device_class_hid_activate</td>
<td>ux_device_class_hid_deactivate</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Hid Activate Event.</td>
<td>This event represents a USBX Device Class Hid Deactivate Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Class Hid Descriptor Send</th>
<th>Device Class Hid Event Get</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td>ux_device_class_hid_descriptor_send</td>
<td>ux_device_class_hid_event_get</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Hid Descriptor Send Event.</td>
<td>This event represents a USBX Device Class Hid Event Get Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Descriptor type.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Request index.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Device Class Hid Event Set</strong></td>
<td><strong>Device Class Hid Report Get</strong></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td><strong>ux_device_class_hid_event_set</strong></td>
<td><strong>ux_device_class_hid_report_get</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Hid Event Set.</td>
<td>This event represents a USBX Device Class Hid Report Get Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Descriptor type.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Request index.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Device Class Hid Report Set</strong></th>
<th><strong>Device Class Pima Activate</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td><strong>ux_device_class_hid_report_set</strong></td>
<td><strong>ux_device_class_pima_activate</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Hid Report Set Event.</td>
<td>This event represents a USBX Device Class Pima Activate Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Descriptor type.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Request index.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Device Class Pima Deactivate</strong></th>
<th><strong>Device Class Pima Device Info Send</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td><strong>ux_device_class_pima_deactivate</strong></td>
<td><strong>ux_device_class_pima_device_info_send</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Pima Deactivate Event.</td>
<td>This event represents a USBX Device Class Pima Device Info Send Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Device Class Pima Event Get</td>
<td>Device Class Pima Event Set</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ux_device_class_pima_event_get</td>
<td>ux_device_class_pima_event_set</td>
</tr>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
</tr>
</tbody>
</table>
This event represents a USBX Device Class Pima Event Get Event. This event represents a USBX Device Class Pima Event Set Event. |
| Information Fields | Information Fields |
Info Field 1: Class Instance. | Info Field 1: Class Instance. |
Info Field 2: Pima event. | Info Field 2: Pima event. |
Info Field 3: Not used. | Info Field 3: Not used. |
Info Field 4: Not used. | Info Field 4: Not used. |

<table>
<thead>
<tr>
<th>Device Class Pima Object Add</th>
<th>Device Class Pima Object Data Get</th>
</tr>
</thead>
<tbody>
<tr>
<td>ux_device_class_pima_object_add</td>
<td>ux_device_class_pima_object_data_get</td>
</tr>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
</tr>
</tbody>
</table>
This event represents a USBX Device Class Pima Object Add Event. This event represents a USBX Device Class Pima Object Data Get Event. |
| Information Fields | Information Fields |
Info Field 1: Class Instance. | Info Field 1: Class Instance. |
Info Field 2: Object handle. | Info Field 2: Object handle. |
Info Field 3: Not used. | Info Field 3: Not used. |
Info Field 4: Not used. | Info Field 4: Not used. |

<table>
<thead>
<tr>
<th>Device Class Pima Object Data Send</th>
<th>Device Class Pima Object Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>ux_device_class_pima_object_data_send</td>
<td>ux_device_class_pima_object_delete</td>
</tr>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
</tr>
</tbody>
</table>
This event represents a USBX Device Class Pima Object Data Send Event. This event represents a USBX Device Class Pima Object Delete Event. |
| Information Fields | Information Fields |
Info Field 1: Class Instance. | Info Field 1: Class Instance. |
Info Field 2: Object handle. | Info Field 2: Object handle. |
Info Field 3: Not used. | Info Field 3: Not used. |
Info Field 4: Not used. | Info Field 4: Not used. |
<table>
<thead>
<tr>
<th>Device Class Pima Object Handles Send</th>
<th>Device Class Pima Object Info Get</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ux_device_class_pima_object_handles_send</strong></td>
<td><strong>ux_device_class_pima_object_info_get</strong></td>
</tr>
<tr>
<td><img src="image1" alt="Icon" /></td>
<td><img src="image2" alt="Icon" /></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Pima Object Handles Send Event.</td>
<td>This event represents a USBX Device Class Pima Object Info Get Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Storage ID.</td>
<td>Info Field 2: Object handle.</td>
</tr>
<tr>
<td>Info Field 3: Object format code.</td>
<td>Info Field 3: Object format code.</td>
</tr>
<tr>
<td>Info Field 4: Object association.</td>
<td>Info Field 4: Object associate.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Class Pima Object Info Send</th>
<th>Device Class Pima Objects NumberSend</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ux_device_class_pima_object_info_send</strong></td>
<td><strong>ux_device_class_pima_objects_number_send</strong></td>
</tr>
<tr>
<td><img src="image3" alt="Icon" /></td>
<td><img src="image4" alt="Icon" /></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Pima Object Info Send Event.</td>
<td>This event represents a USBX Device Class Pima Object Number Send event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Storage ID.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Object format code.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Object associate.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Class Pima Partial Object Data Get</th>
<th>Device Class Pima Response Send</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ux_device_class_pima_partial_object_data_get</strong></td>
<td><strong>ux_device_class_pima_response_send</strong></td>
</tr>
<tr>
<td><img src="image5" alt="Icon" /></td>
<td><img src="image6" alt="Icon" /></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Pima Partial Object Data Get Event.</td>
<td>This event represents a USBX Device Class Pima Response Send Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td>Info Field 2: Object handle.</td>
<td>Info Field 2: Response code.</td>
</tr>
<tr>
<td>Info Field 3: Offset requested.</td>
<td>Info Field 3: Number parameter.</td>
</tr>
<tr>
<td>Info Field 4: Length requested.</td>
<td>Info Field 4: Pima parameter 1.</td>
</tr>
<tr>
<td>Device Class Pima Storage Id Send</td>
<td>Device Class Pima Storage Info Send</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Pima Storage Id Send Event.</td>
<td>This event represents a USBX Device Class Pima Storage Info Send Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Class Rndis Activate</th>
<th>Device Class Rndis Deactivate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Rndis Activate Event.</td>
<td>This event represents a USBX Device Class Rndis Deactivate Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Class Rndis Message Keep Alive</th>
<th>Device Class Rndis Message Query</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Icon</strong></td>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Rndis Message Keep Alive Event.</td>
<td>This event represents a USBX Device Class Rndis Message Query Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Rndis OID.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Device Class Rndis Message Reset</td>
<td>Device Class Rndis Message Set</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Rndis Message Reset Event.</td>
<td>This event represents a USBX Device Class Rndis Message Set Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Rndis OID.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Class Rndis Packet Receive</th>
<th>Device Class Rndis Packet Transmit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Rndis Packet Receive Event.</td>
<td>This event represents a USBX Device Class Rndis Packet Transmit Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Class Storage Activate</th>
<th>Device Class Storage Deactivate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Device Class Storage Activate Event.</td>
<td>This event represents a USBX Device Class Storage Deactivate Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class Instance.</td>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>
### Device Class Storage Format

**ux_device_class_storage_format**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Device Class Storage Format Event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Lun.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

### Device Class Storage Inquiry

**ux_device_class_storage_inquiry**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Device Class Storage Inquiry Event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Lun.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

### Device Class Storage Mode Select

**ux_device_class_storage_mode_select**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Device Class Storage Mode Select Event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Lun.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

### Device Class Storage Mode Sense

**ux_device_class_storage_mode_sense**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Device Class Storage Mode Sense Event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Lun.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

### Device Class Storage Prevent Allow Media Removal

**ux_device_class_storage_prevent_allow_media_removal**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Device Class Storage Prevent Allow Media Removal Event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Lun.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

### Device Class Storage Read

**ux_device_class_storage_read**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Device Class Storage Read Event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info Field 1: Class Instance.</td>
</tr>
<tr>
<td>Info Field 2: Lun.</td>
</tr>
<tr>
<td>Info Field 3: Sector.</td>
</tr>
<tr>
<td>Info Field 4: Number sectors.</td>
</tr>
<tr>
<td>Event Type</td>
</tr>
<tr>
<td>--------------------------------</td>
</tr>
</tbody>
</table>
| Device Class Storage Read Capacity | ux_device_class_storage_read_capacity | This event represents a USBX Device Class Storage Read Capacity Event.       |      | Info Field 1: Class Instance.  
Info Field 2: Lun.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Device Class Storage Read Format Capacity | ux_device_class_storage_read_format_capacity | This event represents a USBX Device Class Storage Read Format Capacity Event. |      | Info Field 1: Class Instance.  
Info Field 2: Lun.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Device Class Storage Read TOC    | ux_device_class_storage_read_toc   | This event represents a USBX Device Class Storage Read TOC Event.            |      | Info Field 1: Class Instance.  
Info Field 2: Lun.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Device Class Storage Request Sense | ux_device_class_storage_request_sense | This event represents a USBX Device Class Storage Request Sense Event.       |      | Info Field 1: Class Instance.  
Info Field 2: Lun.  
Info Field 3: Sense key.  
Info Field 4: Code. |
| Device Class Storage Start Stop  | ux_device_class_storage_start_stop | This event represents a USBX Device Class Storage Start Stop Event.          |      | Info Field 1: Class Instance.  
Info Field 2: Lun.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Device Class Storage Test Ready  | ux_device_class_storage_test_ready | This event represents a USBX Device Class Storage Test Ready Event.          |      | Info Field 1: Class Instance.  
Info Field 2: Lun.  
Info Field 3: Not used.  
Info Field 4: Not used. |
### Device Class Storage Verify

**Description**
This event represents a USBX Device Class Storage Verify Event.

**Information Fields**
- Info Field 1: Class Instance.
- Info Field 2: Lun.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Device Class Storage Write

**Description**
This event represents a USBX Device Class Storage Write Event.

**Information Fields**
- Info Field 1: Class Instance.
- Info Field 2: Lun.
- Info Field 3: Sector.
- Info Field 4: Number sectors.

### Device Stack Alternate Setting Get

**Description**
This event represents a USBX Device Stack Alternate Setting Get Event.

**Information Fields**
- Info Field 1: Interface value.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Device Stack Alternate Setting Set

**Description**
This event represents a USBX Device Stack Alternate Setting Set Event.

**Information Fields**
- Info Field 1: Interface value.
- Info Field 2: Alternate setting value.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Device Stack Class Register

**Description**
This event represents a USBX Device Stack Class Register Event.

**Information Fields**
- Info Field 1: Class name.
- Info Field 2: Interface number.
- Info Field 3: Parameter.
- Info Field 4: Not used.

### Device Stack Clear Feature

**Description**
This event represents a USBX Device Stack Clear Feature Event.

**Information Fields**
- Info Field 1: Request type.
- Info Field 2: Request value.
- Info Field 3: Request index.
- Info Field 4: Not used.
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Function Call</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Stack Configuration Get</td>
<td>ux_device_stack_configuration_get</td>
<td>![Icon]</td>
<td>This event represents a USBX Device Stack Configuration Get Event.</td>
<td>Info Field 1: Configuration value.</td>
</tr>
<tr>
<td>Device Stack Configuration Set</td>
<td>ux_device_stack_configuration_set</td>
<td>![Icon]</td>
<td>This event represents a USBX Device Stack Configuration Set Event.</td>
<td>Info Field 1: Configuration value.</td>
</tr>
<tr>
<td>Device Stack Connect</td>
<td>ux_device_stack_connect</td>
<td>![Icon]</td>
<td>This event represents a USBX Device Stack Descriptor Send Event.</td>
<td>Info Field 1: Not used.</td>
</tr>
<tr>
<td>Device Stack Descriptor Send</td>
<td>ux_device_stack_descriptor_send</td>
<td>![Icon]</td>
<td>This event represents a USBX Device Stack Descriptor Send Event.</td>
<td>Info Field 1: Descriptor type.</td>
</tr>
<tr>
<td>Device Stack Disconnect</td>
<td>ux_device_stack_disconnect</td>
<td>![Icon]</td>
<td>This event represents a USBX Device Stack Disconnect Event.</td>
<td>Info Field 1: Device.</td>
</tr>
<tr>
<td>Device Stack Endpoint Stall</td>
<td>ux_device_stack_endpoint_stall</td>
<td>![Icon]</td>
<td>This event represents a USBX Device Stack Endpoint Stall Event.</td>
<td>Info Field 1: Endpoint.</td>
</tr>
</tbody>
</table>
### Device Stack Get Status

**Icon**

**ux_device_stack_get_status**

**Description**
This event represents a USBX Device Stack Get Status Event.

**Information Fields**
- Info Field 1: Not used.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Device Stack Host Wakeup

**Icon**

**ux_device_stack_host_wakeup**

**Description**
This event represents a USBX Device Stack Host Wakeup Event.

**Information Fields**
- Info Field 1: Not used.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Device Stack Initialize

**Icon**

**ux_device_stack_initialize**

**Description**
This event represents a USBX Device Stack Initialize Event.

**Information Fields**
- Info Field 1: Not used.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Device Stack Interface Delete

**Icon**

**ux_device_stack_interface_delete**

**Description**
This event represents a USBX Device Stack Interface Delete Event.

**Information Fields**
- Info Field 1: Interface.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Device Stack Interface Get

**Icon**

**ux_device_stack_interface_get**

**Description**
This event represents a USBX Device Stack Interface Get Event.

**Information Fields**
- Info Field 1: Interface value.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Device Stack Interface Set

**Icon**

**ux_device_stack_interface_set**

**Description**
This event represents a USBX Device Stack Interface Set Event.

**Information Fields**
- Info Field 1: Request value.
- Info Field 2: Request index.
- Info Field 3: Not used.
- Info Field 4: Not used.
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Device Stack Set Feature         | ux_device_stack_set_feature | This event represents a USBX Device Stack Set Feature Event. | Info Field 1: Request value. 
Info Field 2: Request index. 
Info Field 3: Not used. 
Info Field 4: Not used. |
| Device Stack Transfer Abort      | ux_device_stack_transfer_abort | This event represents a USBX Device Stack Transfer Abort Event. | Info Field 1: Transfer request. 
Info Field 2: Completion code. 
Info Field 3: Not used. 
Info Field 4: Not used. |
| Device Stack Transfer All Request Abort | ux_device_stack_transfer_all_request_abort | This event represents a USBX Device Stack Transfer All Request Abort Event. | Info Field 1: Endpoint. 
Info Field 2: Completion code. 
Info Field 3: Not used. 
Info Field 4: Not used. |
| Device Stack Transfer Request    | ux_device_stack_transfer_request | This event represents a USBX Device Stack Transfer Request Event. | Info Field 1: Transfer request. 
Info Field 2: Not used. 
Info Field 3: Not used. 
Info Field 4: Not used. |
| Host Class Asix Activate         | ux_host_class_asix_activate | This event represents a USBX Host Class Asix Activate. | Info Field 1: Class Instance. 
Info Field 2: Not used. 
Info Field 3: Not used. 
Info Field 4: Not used. |
| Host Class Asix Deactivate       | ux_host_class_asix_deactivate | This event represents a USBX Host Class Asix Deactivate Event. | Info Field 1: Class Instance. 
Info Field 2: Not used. 
Info Field 3: Not used. 
Info Field 4: Not used. |
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Function Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Host Class Asix Interrupt Notification | ux_host_class_asix_interrupt_notification | ![Icon](image) | This event represents a USBX Host Class Asix Interrupt Notification Event. | Info Field 1: Class Instance.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Host Class Asix Read | ux_host_class_asix_read | ![Icon](image) | This event represents a USBX Host Class Asix Read Event. | Info Field 1: Class Instance.  
Info Field 2: Data Pointer.  
Info Field 3: Requested Length.  
Info Field 4: Not used. |
| Host Class Asix Write | ux_host_class_asix_write | ![Icon](image) | This event represents a USBX Host Class Asix Write Event. | Info Field 1: Class Instance.  
Info Field 2: Data Pointer.  
Info Field 3: Requested Length.  
Info Field 4: Not used. |
| Host Class Audio Activate | ux_host_class_audio_activate | ![Icon](image) | This event represents a USBX Host Class Audio Activate Event. | Info Field 1: Class Instance.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Host Class Audio Control Value Get | ux_host_class_audio_control_value_get | ![Icon](image) | This event represents a USBX Host Class Audio Control Value Get Event. | Info Field 1: Class Instance.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Host Class Audio Control Value Set | ux_host_class_audio_control_value_set | ![Icon](image) | This event represents an internal NetX I/O driver deferred processing event. | Info Field 1: Class Instance.  
Info Field 2: Audio Control.  
Info Field 3: Not used.  
Info Field 4: Not used. |
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Event Code</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Host Class Audio Deactivate                    | ux_host_class_audio_deactivate                  | 🟢   | This event represents a USBX Host Class Audio Deactivate Event.               | Info Field 1: Class instance.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Host Class Audio Read                           | ux_host_class_audio_read                        | 🟢   | This event represents a USBX Host Class Audio Read Event.                    | Info Field 1: Class instance.  
Info Field 2: Data pointer.  
Info Field 3: Requested length.  
Info Field 4: Not used. |
| Host Class Audio Streaming Sampling Get         | ux_host_class_audio_streaming_sampling_get      | 🟢   | This event represents a USBX Host Class Audio Streaming Sampling Get Event.  | Info Field 1: Class instance.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Host Class Audio Streaming Sampling Set         | ux_host_class_audio_streaming_sampling_set      | 🟢   | This event represents a USBX Host Class Audio Streaming Sampling Set Event.  | Info Field 1: Class instance.  
Info Field 2: Audio Sampling.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Host Class Audio Write                          | ux_host_class_audio_write                       | 🟢   | This event represents a USBX Host Class Audio Write Event.                   | Info Field 1: Class instance.  
Info Field 2: Data pointer.  
Info Field 3: Requested length.  
Info Field 4: Not used. |
| Host Class Cdc Acm Activate                     | ux_host_class_cdc_acm_activate                  | 🟢   | This event represents a USBX Host Class Cdc Acm Activate Event.              | Info Field 1: Class instance.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Function Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Class Cdc Acm Deactivate</td>
<td>ux_host_class_cdc_acm_deactivate</td>
<td></td>
<td>This event represents a USBX Host Class Cdc Acm Deactivate Event.</td>
<td>Info Field 1: Class instance. Info Field 2: Not used. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Class Cdc Acm IoCtl Abort In Pipe</td>
<td>ux_host_class_cdc_acm_ioctl_abort_in_pipe</td>
<td></td>
<td>This event represents a USBX Host Class Cdc Acm IoCtl Abort In Pipe Event.</td>
<td>Info Field 1: Class instance. Info Field 2: Endpoint. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Class Cdc Acm IoCtl Abort Out Pipe</td>
<td>ux_host_class_cdc_acm_ioctl_abort_out_pipe</td>
<td></td>
<td>This event represents a USBX Host Class Cdc Acm IoCtl Abort Out Pipe Event.</td>
<td>Info Field 1: Class instance. Info Field 2: Endpoint. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Class Cdc Acm IoCtl Get Device Status</td>
<td>ux_host_class_cdc_acm_ioctl_get_device_status</td>
<td></td>
<td>This event represents a USBX Host Class Cdc Acm IoCtl Get Device Status Event.</td>
<td>Info Field 1: Class instance. Info Field 2: Device status. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Class Cdc Acm IoCtl Get Line Coding</td>
<td>ux_host_class_cdc_acm_ioctl_get_line_coding</td>
<td></td>
<td>This event represents a USBX Host Class Cdc Acm IoCtl Get Line Coding Event.</td>
<td>Info Field 1: Class instance. Info Field 2: Parameter. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Class Cdc Acm IoCtl Notification Callback</td>
<td>ux_host_class_cdc_acm_ioctl_notification_callback</td>
<td></td>
<td>This event represents a USBX Host Class Cdc Acm IoCtl Notification Callback Event.</td>
<td>Info Field 1: Class instance. Info Field 2: Parameter. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Event Description</td>
<td>Information Fields</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This event represents a USBX Host Class Cdc Acm Ioctl Send Break Event.</td>
<td>Info Field 1: Class instance.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 2: Parameter.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 3: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 4: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This event represents a USBX Host Class Cdc Acm Ioctl Set Line Coding Event.</td>
<td>Info Field 1: Class instance.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 2: Parameter.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 3: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 4: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This event represents a USBX Host Class Cdc Acm Ioctl Set Line State Event.</td>
<td>Info Field 1: Class instance.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 2: Parameter.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 3: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 4: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This event represents a USBX Host Class Cdc Acm Read Event.</td>
<td>Info Field 1: Class instance.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 2: Data pointer.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 3: Requested Length.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 4: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This event represents a USBX Host Class Cdc Acm Reception Start Event.</td>
<td>Info Field 1: Class instance.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 2: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 3: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 4: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This event represents a USBX Host Class Cdc Acm Reception Stop Event.</td>
<td>Info Field 1: Class instance.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 2: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 3: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info Field 4: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host Class Cdc Acm Write</td>
<td>Host Class Dpump Activate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ux_host_class_cdc_acm_write</td>
<td>ux_host_class_dpump_activate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Icon</td>
<td>Icon</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This event represents a USBX Host Class Cdc Acm Write Event.</td>
<td>This event represents a USBX Host Class Dpump Activate Event.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information Fields</td>
<td>Information Fields</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Info Field 1: Class instance.</td>
<td>Info Field 1: Class instance.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Info Field 2: Data pointer.</td>
<td>Info Field 2: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Info Field 3: Requested Length.</td>
<td>Info Field 3: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host Class Dpump Deactivate</th>
<th>Host Class Dpump Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>ux_host_class_dpump_deactivate</td>
<td>ux_host_class_dpump_read</td>
</tr>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
</tr>
<tr>
<td>This event represents a USBX Host Class Dpump Deactivate Event.</td>
<td>This event represents a USBX Host Class Dpump Read Event.</td>
</tr>
<tr>
<td>Information Fields</td>
<td>Information Fields</td>
</tr>
<tr>
<td>Info Field 1: Class instance.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Requested length.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host Class Dpump Write</th>
<th>Host Class Hid Activate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ux_host_class_dpump_write</td>
<td>ux_host_class_hid_activate</td>
</tr>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
</tr>
<tr>
<td>This event represents a USBX Host Class Dpump Write Event.</td>
<td>This event represents a USBX Host Class Hid Activate Event.</td>
</tr>
<tr>
<td>Information Fields</td>
<td>Information Fields</td>
</tr>
<tr>
<td>Info Field 1: Class instance.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td>Info Field 2: Data pointer.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Requested length.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>
### Host Class Hid Client Register

**ux_host_class_hid_client_register**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Host Class Hid Client Register Event.</td>
</tr>
</tbody>
</table>

**Information Fields**
- Info Field 1: Hid client name.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Hid Deactivate

**ux_host_class_hid_deactivate**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Host Class Hid Deactivate Event.</td>
</tr>
</tbody>
</table>

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Hid Idle Get

**ux_host_class_hid_idle_get**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Host Class Hid Idle Get Event.</td>
</tr>
</tbody>
</table>

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Hid Idle Set

**ux_host_class_hid_idle_set**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Host Class Hid Idle Set Event.</td>
</tr>
</tbody>
</table>

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Hid Keyboard Activate

**ux_host_class_hid_keyboard_activate**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Host Class Hid Keyboard Activate Event.</td>
</tr>
</tbody>
</table>

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Hid client instance.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Hid Keyboard Deactivate

**ux_host_class_hid_keyboard_deactivate**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This event represents a USBX Host Class Hid Keyboard Deactivate Event.</td>
</tr>
</tbody>
</table>

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Hid client instance.
- Info Field 3: Not used.
- Info Field 4: Not used.
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Function Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| Host Class Hid Mouse Activate  | ux_host_class_hid_mouse_activate|      | This event represents a USBX Host Class Hid Mouse Activate Event.              | Info Field 1: Class instance.  
|                                |                                 |      |                                                                             | Info Field 2: Hid client instance.  
|                                |                                 |      |                                                                             | Info Field 3: Not used.  
|                                |                                 |      |                                                                             | Info Field 4: Not used.  |
| Host Class Hid Mouse Deactivate| ux_host_class_hid_mouse_deactivate |      | This event represents a USBX Host Class Hid Mouse Deactivate Event.            | Info Field 1: Class instance.  
|                                |                                 |      |                                                                             | Info Field 2: Hid client instance.  
|                                |                                 |      |                                                                             | Info Field 3: Not used.  
|                                |                                 |      |                                                                             | Info Field 4: Not used.  |
| Host Class Hid Remote Control Activate | ux_host_class_hid_remote_control_activate |      | This event represents a USBX Host Class Hid Remote Control Activate Event.     | Info Field 1: Class instance.  
|                                |                                 |      |                                                                             | Info Field 2: Hid client instance.  
|                                |                                 |      |                                                                             | Info Field 3: Not used.  
|                                |                                 |      |                                                                             | Info Field 4: Not used.  |
| Host Class Hid Remote Control Deactivate | ux_host_class_hid_remote_control_deactivate |      | This event represents a USBX Host Class Hid Remote Control Deactivate Event.    | Info Field 1: Class instance.  
|                                |                                 |      |                                                                             | Info Field 2: Hid client instance.  
|                                |                                 |      |                                                                             | Info Field 3: Not used.  
|                                |                                 |      |                                                                             | Info Field 4: Not used.  |
| Host Class Hid Report Get      | ux_host_class_hid_report_get     |      | This event represents a USBX Host Class Hid Report Get.                       | Info Field 1: Class instance.  
|                                |                                 |      |                                                                             | Info Field 2: Client report.    
|                                |                                 |      |                                                                             | Info Field 3: Not used.        
|                                |                                 |      |                                                                             | Info Field 4: Not used.        |
| Host Class Hid Report Set      | ux_host_class_hid_report_set     |      | This event represents a USBX Host Class Hid Report Set.                       | Info Field 1: Class instance.  
|                                |                                 |      |                                                                             | Info Field 2: Client report.    
|                                |                                 |      |                                                                             | Info Field 3: Not used.        
|                                |                                 |      |                                                                             | Info Field 4: Not used.        |
### Host Class Hub Activate

**Icon**
ux_host_class_hub_activate

**Description**
This event represents a USBX Host Class Hub Activate Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Hub Change Detect

**Icon**
ux_host_class_hub_change_detect

**Description**
This event represents a USBX Host Class Hub Change Detect Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Hub Deactivate

**Icon**
ux_host_class_hub_deactivate

**Description**
This event represents a USBX Host Class Hub Deactivate Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Hub Port Change Connection Process

**Icon**
ux_host_class_hub_port_change_connection_process

**Description**
This event represents a USBX Host Class Hub Port Change Connection Process Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Port.
- Info Field 3: Port status.
- Info Field 4: Not used.

### Host Class Hub Port Change Enable Process

**Icon**
ux_host_class_hub_port_change_enable_process

**Description**
This event represents a USBX Host Class Hub Port Change Enable Process Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Port.
- Info Field 3: Port status.
- Info Field 4: Not used.

### Host Class Hub Port Change Over Current Process

**Icon**
ux_host_class_hub_port_change_over_current_process

**Description**
This event represents allocating a packet via nx_packet_allocate.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Port.
- Info Field 3: Port status.
- Info Field 4: Not used.
<table>
<thead>
<tr>
<th>Host Class Hub Port Change Reset Process</th>
<th>Host Class Hub Port Change Suspend Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>ux_host_class_hub_port_change_reset_process</td>
<td>ux_host_class_hub_port_change_suspend_process</td>
</tr>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Host Class Hub Port Change Reset Process Event.</td>
<td>This event represents a USBX Host Class Hub Port Change Suspend Process Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Hub.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td>Info Field 2: Port.</td>
<td>Info Field 2: Port.</td>
</tr>
<tr>
<td>Info Field 3: Port status.</td>
<td>Info Field 3: Port status.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host Class Pima Activate</th>
<th>Host Class Pima Deactivate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ux_host_class_pima_activate</td>
<td>ux_host_class_pima_deactivate</td>
</tr>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Host Class Pima Activate Event.</td>
<td>This event represents a USBX Host Class Pima Deactivate Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class instance.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host Class Pima Device Info Get</th>
<th>Host Class Pima Device Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ux_host_class_pima_device_info_get</td>
<td>ux_host_class_pima_device_reset</td>
</tr>
<tr>
<td>Icon</td>
<td>Icon</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>This event represents a USBX Host Class Pima Device Info Get Event.</td>
<td>This event represents a USBX Host Class Pima Device Reset Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class instance.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td>Info Field 2: Pima device.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>
### Host Class Pima Notification
- **Icon**: ux_host_class_pima_notification
- **Description**: This event represents a USBX Host Class Pima Notification Event.
- **Information Fields**
  - Info Field 1: Class instance.
  - Info Field 2: Event code.
  - Info Field 3: Transaction ID.
  - Info Field 4: Parameter1.

### Host Class Pima Number Objects Get
- **Icon**: ux_host_class_pima_num_objects_get
- **Description**: This event represents a USBX Host Class Pima Number Objects Get Event.
- **Information Fields**
  - Info Field 1: Class instance.
  - Info Field 2: Not used.
  - Info Field 3: Not used.
  - Info Field 4: Not used.

### Host Class Pima Object Close
- **Icon**: ux_host_class_pima_object_close
- **Description**: This event represents a USBX Host Class Pima Object Close Event.
- **Information Fields**
  - Info Field 1: Class instance.
  - Info Field 2: Object.
  - Info Field 3: Not used.
  - Info Field 4: Not used.

### Host Class Pima Object Copy
- **Icon**: ux_host_class_pima_object_copy
- **Description**: This event represents a USBX Host Class Pima Object Copy Event.
- **Information Fields**
  - Info Field 1: Class instance.
  - Info Field 2: Object handle.
  - Info Field 3: Not used.
  - Info Field 4: Not used.

### Host Class Pima Object Delete
- **Icon**: ux_host_class_pima_object_delete
- **Description**: This event represents a USBX Host Class Pima Object Delete Event.
- **Information Fields**
  - Info Field 1: Class instance.
  - Info Field 2: Object handle.
  - Info Field 3: Not used.
  - Info Field 4: Not used.

### Host Class Pima Object Get
- **Icon**: ux_host_class_pima_object_get
- **Description**: This event represents getting RARP information via nx_rarp_info_get.
- **Information Fields**
  - Info Field 1: Class instance.
  - Info Field 2: Object handle.
  - Info Field 3: Object.
  - Info Field 4: Not used.
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Function Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| **Host Class Pima Object Info Get** | `ux_host_class_pima_object_info_get` | ![Icon] | This event represents a USBX Host Class Pima Object Info Get Event.         | Info Field 1: Class instance.  
Info Field 2: Object handle.  
Info Field 3: Object.  
Info Field 4: Not used. |
| **Host Class Pima Object Info Send** | `ux_host_class_pima_object_info_send` | ![Icon] | This event represents a USBX Host Class Pima Object Info Send Event.        | Info Field 1: Class instance.  
Info Field 2: Object.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **Host Class Pima Object Move**   | `ux_host_class_pima_object_move`      | ![Icon] | This event represents a USBX Host Class Pima Object Move Event.             | Info Field 1: Class instance.  
Info Field 2: Object handle.  
Info Field 3: Object.  
Info Field 4: Not used. |
| **Host Class Pima Object Send**   | `ux_host_class_pima_object_send`      | ![Icon] | This event represents a USBX Host Class Pima Object Send Event.             | Info Field 1: Class instance.  
Info Field 2: Object.  
Info Field 3: Object buffer.  
Info Field 4: Object length. |
| **Host Class Pima Object Transfer Abort** | `ux_host_class_pima_object_transfer_abort` | ![Icon] | This event represents a USBX Host Class Pima Object Transfer Abort Event.    | Info Field 1: Class instance.  
Info Field 2: Object handle.  
Info Field 3: Object.  
Info Field 4: Not used. |
| **Host Class Pima Read**          | `ux_host_class_pima_read`              | ![Icon] | This event represents a USBX Host Class Pima Read Event.                    | Info Field 1: Class instance.  
Info Field 2: Data pointer.  
Info Field 3: Data length.  
Info Field 4: Not used. |
<table>
<thead>
<tr>
<th>Event</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
</table>
| **Host Class Pima Request Cancel**         | ![Image] | This event represents a USBX Host Class Pima Request Cancel Event.                           | Info Field 1: Class instance.            
|                                            |      |                                                                                               | Info Field 2: Not used.                                                            
|                                            |      |                                                                                               | Info Field 3: Not used.                                                            
|                                            |      |                                                                                               | Info Field 4: Not used.                                                            |
| **Host Class Pima Session Close**          | ![Image] | This event represents a USBX Host Class Pima Session Close Event.                            | Info Field 1: Class instance.            
|                                            |      |                                                                                               | Info Field 2: Pima session.                                                       
|                                            |      |                                                                                               | Info Field 3: Not used.                                                            
|                                            |      |                                                                                               | Info Field 4: Not used.                                                            |
| **Host Class Pima Session Open**           | ![Image] | This event represents a USBX Host Class Pima Session Open Event.                             | Info Field 1: Class instance.            
|                                            |      |                                                                                               | Info Field 2: Pima session.                                                       
|                                            |      |                                                                                               | Info Field 3: Not used.                                                            
|                                            |      |                                                                                               | Info Field 4: Not used.                                                            |
| **Host Class Pima Storage Ids Get**        | ![Image] | This event represents a USBX Host Class Pima Storage Ids Get Event.                          | Info Field 1: Class instance.            
|                                            |      |                                                                                               | Info Field 2: Storage ID array.                                                    
|                                            |      |                                                                                               | Info Field 3: Storage ID length.                                                   
|                                            |      |                                                                                               | Info Field 4: Not used.                                                            |
| **Host Class Pima Storage Info Get**       | ![Image] | This event represents a USBX Host Class Pima Storage Info Get Event.                         | Info Field 1: Class instance.            
|                                            |      |                                                                                               | Info Field 2: Storage ID.                                                          
|                                            |      |                                                                                               | Info Field 3: Storage.                                                            
|                                            |      |                                                                                               | Info Field 4: Not used.                                                            |
| **Host Class Pima Thumb Get**              | ![Image] | This event represents unaccepting a TCP server connection via nx_tcp_server_socket_unaccept. | Info Field 1: Class instance.            
|                                            |      |                                                                                               | Info Field 2: Object handle.                                                      
|                                            |      |                                                                                               | Info Field 3: Not used.                                                            
<p>|                                            |      |                                                                                               | Info Field 4: Not used.                                                            |</p>
<table>
<thead>
<tr>
<th>Event Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Host Class Pima Write</strong></td>
<td>Info Field 1: Class instance. Info Field 2: Data pointer. Info Field 3: Data length. Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Host Class Printer Activate</strong></td>
<td>Info Field 1: Pointer to IP instance. Info Field 2: Pointer to socket. Info Field 3: Type of service. Info Field 4: Receive window size.</td>
</tr>
<tr>
<td><strong>Host Class Printer Deactivate</strong></td>
<td>Info Field 1: Class instance. Info Field 2: Not used. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Host Class Printer Name Get</strong></td>
<td>Info Field 1: Class instance. Info Field 2: Not used. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Host Class Printer Read</strong></td>
<td>Info Field 1: Class instance. Info Field 2: Data pointer. Info Field 3: Requested length. Info Field 4: Not used.</td>
</tr>
<tr>
<td><strong>Host Class Printer Soft Reset</strong></td>
<td>Info Field 1: Class instance. Info Field 2: Not used. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Class Printer Status Get</td>
<td>Host Class Printer Write</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td><strong>Icon</strong> ux_host_class_printer_status_get</td>
<td><strong>Icon</strong> ux_host_class_printer_write</td>
</tr>
<tr>
<td><strong>Description</strong> This event represents a USBX Host Class Printer Status Get Event.</td>
<td><strong>Description</strong> This event represents a USBX Host Class Printer Write.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class instance.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td>Info Field 2: Printer status.</td>
<td>Info Field 2: Data pointer.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Requested length.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host Class Prolific Activate</th>
<th>Host Class Prolific Deactivate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Icon</strong> ux_host_class_prolific_activate</td>
<td><strong>Icon</strong> ux_host_class_prolific_deactivate</td>
</tr>
<tr>
<td><strong>Description</strong> This event represents a USBX Host Class Prolific Activate Event.</td>
<td><strong>Description</strong> This event represents a USBX Host Class Prolific Deactivate Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class instance.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td>Info Field 2: Not used.</td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host Class Prolific Ioctl Abort In Pipe</th>
<th>Host Class Prolific Ioctl Abort Out Pipe</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Icon</strong> ux_host_class_prolific_ioctl_abort_in_pipe</td>
<td><strong>Icon</strong> ux_host_class_prolific_ioctl_abort_out_pipe</td>
</tr>
<tr>
<td><strong>Description</strong> This event represents a USBX Host Class Prolific Ioctl Abort In Pipe Event.</td>
<td><strong>Description</strong> This event represents a USBX Host Class Prolific Ioctl Abort Out Pipe Event.</td>
</tr>
<tr>
<td><strong>Information Fields</strong></td>
<td><strong>Information Fields</strong></td>
</tr>
<tr>
<td>Info Field 1: Class instance.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td>Info Field 2: Endpoint.</td>
<td>Info Field 2: Endpoint.</td>
</tr>
<tr>
<td>Info Field 3: Not used.</td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td>Info Field 4: Not used.</td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Event Description</td>
<td>Icon</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
</tbody>
</table>
| **Host Class Prolific Ioctl Get Device Status** | ![Icon](image) | Info Field 1: Class instance.  
Info Field 2: Device status.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **Host Class Prolific Ioctl Get Line Coding** | ![Icon](image) | Info Field 1: Class instance.  
Info Field 2: Parameter.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **Host Class Prolific Ioctl Purge** | ![Icon](image) | Info Field 1: Class instance.  
Info Field 2: Parameter.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **Host Class Prolific Ioctl Report Device Status Change** | ![Icon](image) | Info Field 1: Class instance.  
Info Field 2: Parameter.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **Host Class Prolific Ioctl Send Break** | ![Icon](image) | Info Field 1: Class instance.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **Host Class Prolific Ioctl Set Line Coding** | ![Icon](image) | Info Field 1: Class instance.  
Info Field 2: Parameter.  
Info Field 3: Not used.  
Info Field 4: Not used. |
### Host Class Prolific IOCTL Set Line State

**Icon** ux_host_class_prolific_ioctl_set_line_state

**Description**
This event represents a USBX Host Class Prolific IOCTL Set Line State Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Parameter.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Prolific Read

**Icon** ux_host_class_prolific_read

**Description**
This event represents a USBX Host Class Prolific Read Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Data pointer.
- Info Field 3: Requested length.
- Info Field 4: Not used.

### Host Class Prolific Reception Start

**Icon** ux_host_class_prolific_reception_start

**Description**
This event represents a USBX Host Class Prolific Reception Start Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Prolific Reception Stop

**Icon** ux_host_class_prolific_reception_stop

**Description**
This event represents a USBX Host Class Prolific Reception Stop Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Class Prolific Write

**Icon** ux_host_class_prolific_write

**Description**
This event represents a USBX Host Class Prolific Write Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Data pointer.
- Info Field 3: Requested length.
- Info Field 4: Not used.

### Host Class Storage Activate

**Icon** ux_host_class_storage_activate

**Description**
This event represents a USBX Host Class Storage Activate Event.

**Information Fields**
- Info Field 1: Class instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Class Storage Deactivate</td>
<td></td>
<td>This event represents a USBX Host Class Storage Deactivate Event.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host Class Storage Media Capacity Get</td>
<td></td>
<td>This event represents a USBX Host Class Storage Media Capacity Get Event.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host Class Storage Media Format Capacity Get</td>
<td></td>
<td>This event represents a USBX Host Class Storage Media Format Capacity Get Event.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host Class Storage Media Mount</td>
<td></td>
<td>This event represents a USBX Host Class Storage Media Mount Event.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Sector.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host Class Storage Media Open</td>
<td></td>
<td>This event represents a USBX Host Class Storage Media Open Event.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Media.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host Class Storage Media Read</td>
<td></td>
<td>This event represents a USBX Host Class Storage Media Read Event.</td>
<td>Info Field 1: Class instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Sector start.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Sector count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Data pointer.</td>
</tr>
<tr>
<td>Event Name</td>
<td>Description</td>
<td>Information Fields</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>--------------------</td>
<td></td>
</tr>
</tbody>
</table>
| **Host Class Storage Media Write** | This event represents a USBX Host Class Storage Media Write Event. | Info Field 1: Class instance.  
Info Field 2: Sector start.  
Info Field 3: Sector count.  
Info Field 4: Data pointer. |
| **Host Class Storage Request Sense** | This event represents a USBX Host Class Storage Request Sense Event. | Info Field 1: Class instance.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **Host Class Storage Start Stop** | This event represents a USBX Host Class Storage Start Stop Event. | Info Field 1: Class instance.  
Info Field 2: Start stop signal.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **Host Class Storage Unit Ready Test** | This event represents a USBX Host Class Storage Unit Ready Test Event. | Info Field 1: Class instance.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **Host Stack Class Instance Create** | This event represents a USBX Host Stack Class Instance Create Event. | Info Field 1: Class.  
Info Field 2: Class Instance.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| **Host Stack Class Instance Destroy** | This event represents a USBX Host Stack Class Instance Destroy Event. | Info Field 1: Class.  
Info Field 2: Class Instance.  
Info Field 3: Not used.  
Info Field 4: Not used. |
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Icon</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Stack Configuration Delete</td>
<td></td>
<td>This event represents a USBX Host Stack Configuration Delete Event.</td>
<td>Info Field 1: Configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Stack Configuration Enumerate</td>
<td></td>
<td>This event represents a USBX Host Stack Configuration Enumerate Event.</td>
<td>Info Field 1: Device.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Stack Configuration Instance Create</td>
<td></td>
<td>This event represents a USBX Host Stack Configuration Instance Create Event.</td>
<td>Info Field 1: Configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Stack Configuration Instance Delete</td>
<td></td>
<td>This event represents a USBX Host Stack Configuration Instance Delete Event.</td>
<td>Info Field 1: Configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Stack Configuration Set</td>
<td></td>
<td>This event represents a USBX Host Stack Configuration Set Event.</td>
<td>Info Field 1: Configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Stack Device Address Set</td>
<td></td>
<td>This event represents a USBX Host Stack Device Address Set Event.</td>
<td>Info Field 1: Device.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 2: Device Address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 3: Not used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Info Field 4: Not used.</td>
</tr>
<tr>
<td>Event Type</td>
<td>Function Name</td>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------------------</td>
<td>------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Host Stack Device Configuration Get      | ux_host_stack_device_configuration_get | ![Icon] | This event represents a USBX Host Stack Device Configuration Get Event.     | Info Field 1: Device.  
Info Field 2: Configuration.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Host Stack Device Configuration Select   | ux_host_stack_device_configuration_select | ![Icon] | This event represents a USBX Host Stack Device Configuration Select Event.  | Info Field 1: Device.  
Info Field 2: Configuration.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Host Stack Device Descriptor Read        | ux_host_stack_device_descriptor_read | ![Icon] | This event represents a USBX Host Stack Device Descriptor Read Event.        | Info Field 1: Device.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Host Stack Device Get                    | ux_host_stack_device_get   | ![Icon] | This event represents a USBX Host Stack Device Get Event.                    | Info Field 1: Device index.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
| Host Stack Device Remove                 | ux_host_stack_device_remove | ![Icon] | This event represents a USBX Host Stack Device Remove Event.                 | Info Field 1: Hcd.  
Info Field 2: Parent.  
Info Field 3: Port Index.  
Info Field 4: Device. |
| Host Stack Device Resource Free          | ux_host_stack_device_resource_free | ![Icon] | This event represents a USBX Host Stack Device Resource Free Event.          | Info Field 1: Device.  
Info Field 2: Not used.  
Info Field 3: Not used.  
Info Field 4: Not used. |
### Host Stack Endpoint Instance Create
```
ux_host_stack_endpoint_instance_create
```

**Icon**

**Description**
This event represents a USBX Host Stack Endpoint Instance Create Event.

**Information Fields**
- Info Field 1: Device.
- Info Field 2: Endpoint.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Stack Endpoint Instance Delete
```
ux_host_stack_endpoint_instance_delete
```

**Icon**

**Description**
This event represents a USBX Host Stack Endpoint Instance Delete Event.

**Information Fields**
- Info Field 1: Device.
- Info Field 2: Endpoint.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Stack Endpoint Reset
```
ux_host_stack_endpoint_reset
```

**Icon**

**Description**
This event represents a USBX Host Stack Endpoint Reset Event.

**Information Fields**
- Info Field 1: Device.
- Info Field 2: Endpoint.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Stack Endpoint Transfer Abort
```
ux_host_stack_endpoint_transfer_abort
```

**Icon**

**Description**
This event represents a USBX Host Stack Endpoint Transfer Abort Event.

**Information Fields**
- Info Field 1: Endpoint.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Stack Host Controller Register
```
ux_host_stack_hcd_register
```

**Icon**

**Description**
This event represents a USBX Host Stack Host Controller Register.

**Information Fields**
- Info Field 1: Hcd Name.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Stack Initialize
```
ux_host_stack_initialize
```

**Icon**

**Description**
This event represents a USBX Host Stack Initialize Event.

**Information Fields**
- Info Field 1: Not used.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Information Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Stack Interface Endpoint Get</td>
<td>This event represents an internal NetX TCP retry event.</td>
<td>Info Field 1: Interface. Info Field 2: Endpoint index. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Stack Interface Instance Create</td>
<td>This event represents a USBX Host Stack Interface Instance Create Event.</td>
<td>Info Field 1: Interface. Info Field 2: Not used. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Stack Interface Instance Delete</td>
<td>This event represents a USBX Host Stack Interface Instance Delete Event.</td>
<td>Info Field 1: Interface. Info Field 2: Not used. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Stack Interface Setting Select</td>
<td>This event represents a USBX Host Stack Interface Setting Select Event.</td>
<td>Info Field 1: Interface. Info Field 2: Not used. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
<tr>
<td>Host Stack New Configuration Create</td>
<td>This event represents a USBX Host Stack New Configuration Create Event.</td>
<td>Info Field 1: Device. Info Field 2: Configuration. Info Field 3: Not used. Info Field 4: Not used.</td>
</tr>
</tbody>
</table>
### Host Stack New Device Create

**ux_host_stack_new_device_create**

**Icon**

**Description**
This event represents a USBX Host Stack New Device Create Event.

**Information Fields**
- Info Field 1: Hcd.
- Info Field 2: Device owner.
- Info Field 3: Port index.
- Info Field 4: Device.

### Host Stack New Endpoint Create

**ux_host_stack_new_endpoint_create**

**Icon**

**Description**
This event represents a USBX Host Stack New Endpoint Create Event.

**Information Fields**
- Info Field 1: Interface.
- Info Field 2: Endpoint.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Stack Root Hub Change Process

**ux_host_stack_rh_change_process**

**Icon**

**Description**
This event represents a USBX Host Stack Root Hub Change Process.

**Information Fields**
- Info Field 1: Port index.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Stack Root Hub Device Extraction

**ux_host_stack_rh_device_extraction**

**Icon**

**Description**
This event represents a USBX Host Stack Root Hub Device Extraction Event.

**Information Fields**
- Info Field 1: Hcd.
- Info Field 2: Port index.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Stack Root Hub Device Insertion

**ux_host_stack_rh_device_insertion**

**Icon**

**Description**
This event represents a USBX Host Stack Root Hub Device Insertion.

**Information Fields**
- Info Field 1: Hcd.
- Info Field 2: Port index.
- Info Field 3: Not used.
- Info Field 4: Not used.

### Host Stack Transfer Request

**ux_host_stack_transfer_request**

**Icon**

**Description**
This event represents a USBX Host Stack Transfer Request.

**Information Fields**
- Info Field 1: Device.
- Info Field 2: Endpoint.
- Info Field 3: Transfer request.
- Info Field 4: Not used.
## Host Stack Transfer Request Abort

<table>
<thead>
<tr>
<th>Icon</th>
<th>Internal I/O driver get status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**  
This event represents a USBX Host Stack Transfer Request Abort.

**Information Fields**  
- Info Field 1: Device.
- Info Field 2: Endpoint.
- Info Field 3: Transfer request.
- Info Field 4: Not used.

## USB Error

<table>
<thead>
<tr>
<th>Icon</th>
<th>ux_error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**  
This event represents a USBX Error Event.

**Information Fields**  
- Info Field 1: USBX error.
- Info Field 2: Error Name.
- Info Field 3: Not used.
- Info Field 4: Not used.
This chapter contains a description of how to create user-defined events and custom icons and information fields for such events. This chapter includes the following sections:

- Inserting User-Defined Events 222
- Default Display of User-Defined Events 222
- Defining Custom User-Defined Event Icons 224
Inserting User-Defined Events

ThreadX provides the ability for developers to log their own user-defined events, providing even more useful information that can be viewed graphically by TraceX.

User-defined event numbers range from `TX_TRACE_USER_EVENT_START` (4096) through `TX_TRACE_USER_EVENT_END` (65535), inclusive. The placement of the events in the trace buffer is done via the `tx_trace_user_event_insert`, defined in Chapter 5. The following example calls insert two user-defined events into the current trace buffer on the target, namely user-defined event 4096 and event 4098:

```c
tx_trace_user_event_insert(4096, 1, 2, 3, 4);
```
```c
tx_trace_user_event_insert(4098, 0x100, 0x200, 0x300, 0x400);
```

Default Display of User-Defined Events

By default, TraceX displays all user events with a default user-defined Event icon as described in Chapter 6. Figure 10.1 shows the default user-defined event icon for events 202
and 203, which were placed in the event buffer via the previous `tx_trace_user_event_insert` examples.

**FIGURE 10.1**

Detailed information is also available for user-defined Events. Figure 10.2 shows the detailed event information for event 202, which has event number 4096 and shows the specified 4 information fields.

**FIGURE 10.2**
Defining Custom User-Defined Event Icons

TraceX also provides the user the ability to create custom user-defined event icons as well as custom information field labels. This is achieved by adding event icon specifications to the `eltrxcustom.trxc` configuration file, which is located in the `CustomEvents` sub-directory in the main TraceX installation directory. An example directory path is shown in Figure 10.3.

![Figure 10.3](image)

The `eltrxcustom.trxc` custom event configuration file is a simple ASCII text file containing 0 or more custom event definitions. The format of the file is as follows:

```plaintext
//Comments
Start
[custom event definition(s)]
End
```

Each line between Start and End is used to defined a single custom event. TraceX provides a template version of this file with no custom events defined (nothing between the “Start”
and “End” labels). The format of a custom event definition is as follows:

\[ \text{number}, \text{name}, \text{abbreviation}, \text{top\_color}, \text{bottom\_color}, \text{label1}, \text{label2}, \text{label2}, \text{label4} \]

where:

- **number**: Defines the user-defined event number, between 4096 and 65535, inclusive.
- **name**: Defines the logical name for the user-defined event.
- **abbreviation**: Defines the 2-letter user-defined event abbreviation.
- **top\_color**: Defines the RGB value for the top-half of the icon, which is a three digit number in parenthesis. Some typical RGB definitions are:
  - \text{BLACK} = (0,0,0)
  - \text{WHITE} = (255,255,255)
  - \text{RED} = (255,0,0)
  - \text{GREEN} = (0,255,0)
  - \text{BLUE} = (0,0,255)
  - \text{YELLOW} = (255,255,0)
  - \text{CYAN} = (0,255,255)
  - \text{MAGENTA} = (255,0,255)

Using the RBG specification gives the user a broad range of colors for each user-defined icon. For more information on RBG color definition, please see:

http://en.wikipedia.org/wiki/RGB#Digital_representations

- **bottom\_color**: Defines the RGB value for the bottom-half of the icon, which is a three digit number in parenthesis.
Example definitions for each of the two user-defined events used in this chapter are shown in Figure 10.4. The first definition is for event 4096 at line 5 of the `eltrxcustom.trxc` file. This definition gives user-defined event 4096 the name `First_User_Event`, specifies a two-letter abbreviation of `FE`, makes the top portion of the icon red, the bottom portion of the icon green, and names the information fields as `First_Info1`, `First_Info2`, `First_Info3`, and
**First_Info4.** User-defined event 4098 is defined similarly at line 6 of `eltrxcustom.trxc`.

Since the `eltrxcustom.trxc` file is read by TraceX during initialization, TraceX must be exited and restarted before the custom icon definitions can take effect. Figure 10.5 shows the TraceX display of user-
defined events 202 and 203 with the custom event icons defined in `eltrxcustom.trxc`.

![Image of custom event icons](image)

**FIGURE 10.5**

The additional information in the custom event definition is shown when the event is selected via a double-click, mouse-over, or selection of the current event button. Figure 10.6 shows the double-click selection on event 202. Note that the event name
and information fields all match the sample definition that was added to `eltrxcustom.trxc`.

**FIGURE 10.6**
ThreadX provides built-in event trace support for all ThreadX services, thread state changes, and user-defined events. To use event trace, simply build the ThreadX, NetX, and FileX libraries with `TX_ENABLE_EVENT_TRACE` defined and enable tracing by calling the `tx_trace_enable` function. This chapter describes that process.

- Event Trace Format 232
- Event Trace Control Header 232
  - Control Header ID 233
  - Timer Valid Mask 233
  - Trace Base Address 234
  - Registry Start and End Pointers 234
  - Registry Name Size 234
  - Buffer Start and End Pointers 234
  - Current Buffer Pointer 235
- Event Trace Object Registry 235
  - Object Available Flag 235
  - Object Entry Type 236
  - Object Pointer 237
  - Object Reserved Fields 237
  - Object Parameters 237
  - Object Name 237
- Event Trace Entries 238
  - Thread Pointer 238
  - Thread Priority 239
  - Event ID 239
- Information Fields (1-4) 239
Event Trace Format

The format of the ThreadX event trace buffer is divided into three sections, namely the control header, object registry, and the trace entries. The following describes the general layout of the ThreadX event trace buffer:

Control Header
Object Registry Entry 0
...
Object Register Entry “n”
Event Trace Entry 0
...
Event Trace Entry “n”

Event Trace Control Header

The control header defines the exact layout of the event trace buffer. This includes how many ThreadX objects can be registered as well as how many events can be recorded. In addition, the control header defines where each of the elements of the
The trace buffer resides. The following data structure defines the control header:

```c
typedef struct TX_TRACE_CONTROL_HEADER_STRUCT
{
    ULONG tx_trace_control_header_id;
    ULONG tx_trace_control_header_timer_valid_mask;
    ULONG tx_trace_control_header_trace_base_address;
    ULONG tx_trace_control_header_object_registry_start_pointer;
    USHORT tx_trace_control_header_reserved1;
    USHORT tx_trace_control_header_object_registry_name_size;
    ULONG tx_trace_control_header_object_registry_end_pointer;
    ULONG tx_trace_control_header_buffer_start_pointer;
    ULONG tx_trace_control_header_buffer_end_pointer;
    ULONG tx_trace_control_header_buffer_current_pointer;
    ULONG tx_trace_control_header_reserved2;
    ULONG tx_trace_control_header_reserved3;
    ULONG tx_trace_control_header_reserved4;
} TX_TRACE_CONTROL_HEADER;
```

### Control Header ID

The control header ID consists of the 32-bit HEX value of 0x54585442, which corresponds to the ASCII characters `TXTB`. Since this value is written as a 32-bit unsigned variable, it can also be used to detect the endianness of the event trace buffer. For example, if the value in the first four bytes of memory is 0x54, 0x58, 0x54, 0x42, the event trace buffer was written in big endian format. Otherwise, the event trace buffer was written in little endian format.

### Timer Valid Mask

The timer valid mask defines how many bits of the time-stamp in the actual event trace entries are valid. For example, if the time-stamp source has 16-bits, the value in this field should be 0xFFFF. A 32-bit time-stamp source would have a value of 0xFFFFFFFF. This value is defined by the `TX_TRACE_TIME_MASK` constant in `tx_port.h`. 

---

Express Logic, Inc.
Trace Base Address

The trace buffer base address is the address the application specified as the start of the trace buffer in the `tx_trace_enable` call. This address is maintained for the sole use of the analysis tool to derive buffer-relative offsets for the various elements in the buffer. For example, the buffer relative offset of the current event in the trace buffer is calculated by simple subtraction of the base address from the current event address.

Registry Start and End Pointers

The registry start pointer points to the address of the first object registry entry, while the registry end pointer points to the address immediately following the last register entry. These values are setup during the `tx_trace_enable` processing and are not changed throughout the duration of tracing.

Registry Name Size

The registry name size defines maximum size in bytes for each object name in the registry entry and is defined by the symbol `TX_TRACE_OBJECT_REGISTRY_NAME`. The default value is 32 and is defined in `tx_trace.h`. The object name corresponds to the name given by the application when the object was created. For example, the object registry name for a thread is the name supplied by the application to the `tx_thread_create` call.

Buffer Start and End Pointers

The event trace buffer start pointer points to the address of the first trace entry, while the registry end pointer points to the address immediately following the last trace entry. These values are setup during the `tx_trace_enable` processing and are not changed throughout the duration of tracing.
Current Buffer Pointer

The event trace buffer current pointer points to the address of the oldest trace entry. Since the trace entries are maintained in a circular list, the current buffer pointer is also represents the next trace entry to be written.

Event Trace Object Registry

The event trace object registry contains \( n \) object registry entries that correspond to the objects created by the application. The main purpose of the object registry is for external analysis tools to correlate actual object names with the object addresses of the trace buffer entries. The number of registry entries is specified by the application in the `tx_trace_enable` call.

Each object register entry contains information about a specific ThreadX object previously created by the application. The following data structure defines each object registry entry:

```c
typedef struct TX_TRACE_OBJECT_REGISTRY_ENTRY_STRUCT
{
    UCHAR tx_trace_object_registry_entry_object_available;
    UCHAR tx_trace_object_registry_entry_object_type;
    UCHAR tx_trace_object_registry_entry_object_reserved1;
    UCHAR tx_trace_object_registry_entry_object_reserved2;
    ULONG tx_trace_object_registry_entry_object_pointer;
    ULONG tx_trace_object_registry_entry_object_parameter_1;
    ULONG tx_trace_object_registry_entry_object_parameter_2;
    UCHAR tx_trace_thread_registry_entry_object_name [TX_TRACE_OBJECT_REGISTRY_NAME];
} TX_TRACE_OBJECT_REGISTRY_ENTRY;
```

Object Available Flag

The object available flag is set to 1 if the object registry entry is available. Otherwise, if the value is not 1, the object registry entry is not available. Note
that the entry could still contain valid information even though it is available.

**Object Entry Type**

The object entry type identifies the type of object in this entry. The following is a list of the valid object types:

<table>
<thead>
<tr>
<th>Value</th>
<th>Object Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not Valid</td>
</tr>
<tr>
<td>1</td>
<td>Thread</td>
</tr>
<tr>
<td>2</td>
<td>Timer</td>
</tr>
<tr>
<td>3</td>
<td>Queue</td>
</tr>
<tr>
<td>4</td>
<td>Semaphore</td>
</tr>
<tr>
<td>5</td>
<td>Mutex</td>
</tr>
<tr>
<td>6</td>
<td>Event Flags Group</td>
</tr>
<tr>
<td>7</td>
<td>Block Pool</td>
</tr>
<tr>
<td>8</td>
<td>Byte Pool</td>
</tr>
<tr>
<td>9</td>
<td>Media</td>
</tr>
<tr>
<td>10</td>
<td>File</td>
</tr>
<tr>
<td>11</td>
<td>IP</td>
</tr>
<tr>
<td>12</td>
<td>Packet Pool</td>
</tr>
<tr>
<td>13</td>
<td>TCP Socket</td>
</tr>
<tr>
<td>14</td>
<td>UDP Socket</td>
</tr>
<tr>
<td>15-20</td>
<td>Reserved</td>
</tr>
<tr>
<td>21</td>
<td>USB Host Stack Device</td>
</tr>
<tr>
<td>22</td>
<td>USB Host Stack Interface</td>
</tr>
<tr>
<td>23</td>
<td>USB Host Endpoint</td>
</tr>
<tr>
<td>24</td>
<td>USB Host Class</td>
</tr>
<tr>
<td>25</td>
<td>USB Device</td>
</tr>
<tr>
<td>26</td>
<td>USB Device Interface</td>
</tr>
<tr>
<td>27</td>
<td>USB Device Endpoint</td>
</tr>
<tr>
<td>28</td>
<td>USB Device Class</td>
</tr>
</tbody>
</table>
Object Pointer

The object pointer specifies the object address that is used for accessing the object using the ThreadX API.

Object Reserved Fields

For all objects other than threads, these reserved fields should be 0. For threads, the priority of the thread at the time it is entered into the registry is placed in these two reserved fields.

Object Parameters

The object parameters contain supplemental information about the object. The following describes the supplemental information for each ThreadX object:

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Parameter 1</th>
<th>Parameter 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>Stack Start</td>
<td>Stack Size</td>
</tr>
<tr>
<td>Timer</td>
<td>Initial Ticks</td>
<td>Reschedule Ticks</td>
</tr>
<tr>
<td>Queue</td>
<td>Queue Size</td>
<td>Message Size</td>
</tr>
<tr>
<td>Semaphore</td>
<td>Initial Instances</td>
<td>-</td>
</tr>
<tr>
<td>Mutex</td>
<td>Inheritance Flag</td>
<td>-</td>
</tr>
<tr>
<td>Event Flags Group</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Block Pool</td>
<td>Total Blocks</td>
<td>Block Size</td>
</tr>
<tr>
<td>Byte Pool</td>
<td>Total Bytes</td>
<td>-</td>
</tr>
<tr>
<td>Media</td>
<td>Fat Cache Size</td>
<td>Sector Cache Size</td>
</tr>
<tr>
<td>File</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IP</td>
<td>Stack Start</td>
<td>Stack Size</td>
</tr>
<tr>
<td>Packet Pool</td>
<td>Packet Size</td>
<td>Number of Packets</td>
</tr>
<tr>
<td>TCP Socket</td>
<td>IP address</td>
<td>Window Size</td>
</tr>
<tr>
<td>UDP Socket</td>
<td>IP address</td>
<td>RX Queue Max</td>
</tr>
</tbody>
</table>

Object Name

The object name contains the name of the ThreadX object. The name is the name provided to ThreadX at the time the object was created. By default, the object
name has a maximum of 32 characters. Actual names greater than 32 characters are truncated.

**Event Trace Entries**

The event trace entries are found in the bottom portion of the event trace buffer. The entries are maintained in a circular list, with the current entry pointer pointing to the oldest entry. The number of entries in the list is calculated by the `tx_trace_enable` call.

Each object register entry contains information about a specific ThreadX trace event. The following data structure defines each trace event entry:

```c
typedef struct TX_TRACE_BUFFER_ENTRY_STRUCT
{
    ULONG tx_trace_buffer_entry_thread_pointer;
    ULONG tx_trace_buffer_entry_thread_priority;
    ULONG tx_trace_buffer_entry_event_id;
    ULONG tx_trace_buffer_entry_time_stamp;
    ULONG tx_trace_buffer_entry_information_field_1;
    ULONG tx_trace_buffer_entry_information_field_2;
    ULONG tx_trace_buffer_entry_information_field_3;
    ULONG tx_trace_buffer_entry_information_field_4;
} TX_TRACE_BUFFER_ENTRY;
```

**Thread Pointer**

The thread pointer contains the address of the thread running at the time of this event. If the event occurred during initialization (no thread running), the value of this pointer is 0xF0F0F0F0. If the event occurred during an Interrupt Service Routine (ISR), the value of this pointer is 0xFFFFFFFF. If the entry has not yet been used, the value of this pointer is 0.
Thread Priority

The thread priority field contains the thread priority and preemption-threshold of the thread that was running at the time of this event. If an interrupt context is present (thread pointer is 0xFFFFFFFF), the value of this field is not the priority but instead the value of \_tx\_thread\_current\_ptr at the time of the event. Otherwise, the value of this field is 0.

Event ID

The event ID specifies the event that took place. Valid ThreadX trace event IDs range from 1 through 1024. Values starting at 1025 and above are reserved for user-specific events. Please refer to the tx_trace.h file for the complete definition of ThreadX event IDs.

Information Fields (1-4)

The information fields contain additional information about the specific event. Please refer to the tx_trace.h file for the complete description of the information fields for each of the defined ThreadX event IDs.
This chapter displays a sample `tx_port.h` file.
#ifndef TX_PORT_H
#define TX_PORT_H

/* Determine if the optional ThreadX user define file should be used. */
#endif //TX_INCLUDE_USER_DEFINE_FILE

/* PORT SPECIFIC C INFORMATION                            RELEASE        */
/* tx_port.h                                           ARM7/RVDS       */
/* 5.0          */
/* AUTHOR                                                                */
/* William E. Lamie, Express Logic, Inc.                               */
/* DESCRIPTION                                                           */
/* This file contains data type definitions that make the ThreadX   */
/* real-time kernel function identically on a variety of different   */
/* processor architectures. For example, the size or number of bits   */
/* in an "int" data type vary between microprocessor architectures and */
/* even C compilers for the same microprocessor. ThreadX does not    */
/* directly use native C data types. Instead, ThreadX creates its    */
/* own special types that can be mapped to actual data types by this  */
/* file to guarantee consistency in the interface and functionality.  */
/* RELEASE HISTORY                                                       */
/* DATE              NAME                      DESCRIPTION             */
/* 12-12-2005     William E. Lamie         Initial ARM7 RealView     */
/* Support Version 5.0          */

#ifndef TX_PORT_H
#define TX_PORT_H

/* Determine if the optional ThreadX user define file should be used. */
#endif //TX_INCLUDE_USER_DEFINE_FILE
/* Yes, include the user defines in tx_user.h. The defines in this file may
   alternately be defined on the command line. */
#endif

/* Define compiler library include files. */
#include <stdlib.h>
#include <string.h>

/* Define ThreadX basic types for this port. */
#define VOID                                    void
typedef char                                    CHAR;
typedef unsigned char                           UCHAR;
typedef int                                     INT;
typedef unsigned int                            UINT;
typedef long                                    LONG;
typedef unsigned long                           ULONG;
typedef short                                   SHORT;
typedef unsigned short                          USHORT;

/* Define the priority levels for ThreadX. Legal values range
   from 32 to 1024 and MUST be evenly divisible by 32. */
#ifndef TX_MAX_PRIORITIES
#define TX_MAX_PRIORITIES                       32
#endif

/* Define the minimum stack for a ThreadX thread on this processor. If the size supplied during
   thread creation is less than this value, the thread create call will return an error. */
#ifndef TX_MINIMUM_STACK
#define TX_MINIMUM_STACK                         200 /* Minimum stack size for this port */
#endif

/* Define the system timer thread's default stack size and priority. These are only applicable
   if TX_TIMER_PROCESS_IN_ISR is not defined. */
#ifndef TX_TIMER_THREAD_STACK_SIZE
#define TX_TIMER_THREAD_STACK_SIZE                1024 /* Default timer thread stack size */
#endif

#ifndef TX_TIMER_THREAD_PRIORITY
#define TX_TIMER_THREAD_PRIORITY                  0  /* Default timer thread priority */
#endif

/* Define various constants for the ThreadX ARM port. */
#ifndef TX_ENABLE_FIQ_SUPPORT
#define TX_INT_DISABLE                            0xC0  /* Disable IRQ & FIQ interrupts */
#else
#define TX_INT_DISABLE                            0x80  /* Disable IRQ interrupts */
#endif
#define TX_INT_ENABLE                              0x00  /* Enable IRQ interrupts */

/* Define the clock source for trace event entry time stamp. The following two item are port specific.
   For example, if the time source is at the address 0x0a800024 and is 16-bits in size, the clock
   source constants would be:
*/

#define TX_TRACE_TIME_SOURCE                    ++_tx_trace_simulated_time
#define TX_TRACE_TIME_SOURCE                    *((ULONG *) 0x0a800024)
#define TX_TRACE_TIME_MASK                      0x0000FFFFUL

#ifndef TX_TRACE_TIME_SOURCE
#define TX_TRACE_TIME_SOURCE                    ++_tx_trace_simulated_time
#ifndef TX_TRACE_TIME_MASK
#define TX_TRACE_TIME_MASK 0xFFFFFFFFUL
#endif

/* Define the port specific options for the _tx_build_options variable. This variable indicates
how the ThreadX library was built. */

#ifdef TX_ENABLE_FIQ_SUPPORT
#define TX_FIQ_ENABLED 1
#else
#define TX_FIQ_ENABLED 0
#endif

#ifdef TX_ENABLE_IRQ_NESTING
#define TX_IRQ_NESTING_ENABLED 2
#else
#define TX_IRQ_NESTING_ENABLED 0
#endif

#ifdef TX_ENABLE_FIQ_NESTING
#define TX_FIQ_NESTING_ENABLED 4
#else
#define TX_FIQ_NESTING_ENABLED 0
#endif

#define TX_PORT_SPECIFIC_BUILD_OPTIONS TX_FIQ_ENABLED | TX_IRQ_NESTING_ENABLED |
TX_FIQ_NESTING_ENABLED

/* Define the in-line initialization constant so that modules with in-line
initialization capabilities can prevent their initialization from being
a function call. */

#define TX_INLINE_INITIALIZATION

/* Determine whether or not stack checking is enabled. By default, ThreadX stack checking is
disabled. When the following is defined, ThreadX thread stack checking is enabled. If stack
checking is enabled (TX_ENABLE_STACK_CHECKING is defined), the TX_DISABLE_STACK_FILLING
define is negated, thereby forcing the stack fill which is necessary for the stack checking
logic. */

#ifdef TX_ENABLE_STACK_CHECKING
#undef TX_DISABLE_STACK_FILLING
#endif

/* Define the TX_THREAD control block extensions for this port. The main reason
for the multiple macros is so that backward compatibility can be maintained with
existing ThreadX kernel awareness modules. */

#define TX_THREAD_EXTENSION_0
#define TX_THREAD_EXTENSION_1
#define TX_THREAD_EXTENSION_2
#define TX_THREAD_EXTENSION_3

/* Define the port extensions of the remaining ThreadX objects. */

#define TX_BLOCK_POOL_EXTENSION
#define TX_BYTE_POOL_EXTENSION
#define TX_EVENT_FLAGS_GROUP_EXTENSION
#define TX_MUTEX_EXTENSION
#define TX_QUEUE_EXTENSION
#define TX_SEMAPHORE_EXTENSION
#define TX_TIMER_EXTENSION
/* Define the user extension field of the thread control block. Nothing additional is needed for this port so it is defined as white space. */
#ifndef TX_THREAD_USER_EXTENSION
#define TX_THREAD_USER_EXTENSION
#endif

/* Define the macros for processing extensions in tx_thread_create, tx_thread_delete, tx_thread_shell_entry, and tx_thread_terminate. */
#define TX_THREAD_CREATE_EXTENSION(thread_ptr)
#define TX_THREAD_DELETE_EXTENSION(thread_ptr)
#define TX_THREAD_COMPLETED_EXTENSION(thread_ptr)
#define TX_THREAD_TERMINATED_EXTENSION(thread_ptr)

/* Define the ThreadX object creation extensions for the remaining objects. */
#define TX_BLOCK_POOL_CREATE_EXTENSION(pool_ptr)
#define TX_BYTE_POOL_CREATE_EXTENSION(pool_ptr)
#define TX_EVENT_FLAGS_GROUP_CREATE_EXTENSION(group_ptr)
#define TX_MUTEX_CREATE_EXTENSION(mutex_ptr)
#define TX_QUEUE_CREATE_EXTENSION(queue_ptr)
#define TX_SEMAPHORE_CREATE_EXTENSION(semaphore_ptr)
#define TX_TIMER_CREATE_EXTENSION(timer_ptr)

/* Define the ThreadX object deletion extensions for the remaining objects. */
#define TX_BLOCK_POOL_DELETE_EXTENSION(pool_ptr)
#define TX_BYTE_POOL_DELETE_EXTENSION(pool_ptr)
#define TX_EVENT_FLAGS_GROUP_DELETE_EXTENSION(group_ptr)
#define TX_MUTEX_DELETE_EXTENSION(mutex_ptr)
#define TX_QUEUE_DELETE_EXTENSION(queue_ptr)
#define TX_SEMAPHORE_DELETE_EXTENSION(semaphore_ptr)
#define TX_TIMER_DELETE_EXTENSION(timer_ptr)

/* Define ThreadX interrupt lockout and restore macros for protection on access of critical kernel information. The restore interrupt macro must restore the interrupt posture of the running thread prior to the value present prior to the disable macro. In most cases, the save area macro is used to define a local function save area for the disable and restore macros. */
#ifndef __thumb
#define TX_INTERRUPT_SAVE_AREA                  register unsigned int interrupt_save, temp;
#ifdef TX_ENABLE_FIQ_SUPPORT
#define TX_DISABLE                              __asm
          {
          MRS  interrupt_save, CPSR;
          ORR  temp, interrupt_save, 0xC0;
          MSR  CPSR_c, temp
          }
#else
#define TX_DISABLE                              __asm
          {
          MRS  interrupt_save, CPSR;
          ORR  temp, interrupt_save, 0x80;
          MSR  CPSR_c, temp
          }
#endif
#define TX_RESTORE                              __asm
          {
          MSR CPSR_c, interrupt_save
          }
#else
#define TX_INTERRUPT_SAVE_AREA
#define TX_DISABLE
#define TX_RESTORE
#endif
#else

unsigned int   _tx_thread_interrupt_disable(void);
unsigned int   _tx_thread_interrupt_restore(UINT old_posture);

#define TX_INTERRUPT_SAVE_AREA   unsigned int interrupt_save;
#define TX_DISABLE               interrupt_save = _tx_thread_interrupt_disable();
#define TX_RESTORE               _tx_thread_interrupt_restore(interrupt_save);
#endif

/* Define the interrupt lockout macros for each ThreadX object. */
#define TX_BLOCK_POOL_DISABLE     TX_DISABLE
#define TX_BYTE_POOL_DISABLE      TX_DISABLE
#define TX_EVENT_FLAG_GROUP_DISABLE TX_DISABLE
#define TX_MUTEX_DISABLE          TX_DISABLE
#define TX_QUEUE_DISABLE          TX_DISABLE
#define TX_SEMAPHORE_DISABLE      TX_DISABLE

/* Define the version ID of ThreadX. This may be utilized by the application. */
#ifdef TX_THREAD_INIT
CHAR                _tx_version_id[] =
                    "Copyright (c) 1996-YYYY Express Logic Inc. * ThreadX ARM7/RVDS
                    Version GVVVV.5.0 SN: ZZZZ *");
#else
extern  CHAR        _tx_version_id[];
#endif

#endif
This chapter displays the `tx_trace.h` file.
TRACEX User Guide

/* Include necessary system files. */

#ifndef TX_TRACE_H
#define TX_TRACE_H

/* Determine if tracing is enabled. If not, simply define the in-line trace macros to whitespace. */

#ifndef TX_ENABLE_EVENT_TRACE
#define TX_TRACE_IN_LINE_INSERT(i,a,b,c,d,f)
#define TX_TRACE_OBJECT_UNREGISTER(o)
#define TX_TRACE_OBJECT_REGISTER(t,p,n,a,b)
#define TX_TRACE_INITIALIZE
#endif

#ifndef TX_ENABLE_EVENT_TRACE
#define TX_TRACE_H
#endif

/***************************************************************************/
/*                                           version 5.4                        */
/*                                           priority, resulting in        */
/*                                           the thread's preemption-    */
/*                                           and added logic to insert   */
/*                                           defines for default source, */
/*                                           and structure definitions as well as external references. It is */
/*                                           assumed that tx_api.h and tx_port.h have already been included. */
/*                                           */
/*                                           RELEASE HISTORY                      */
/*                                           */
/*                                           DATE              NAME                      DESCRIPTION             */
/*                                           */
/*                                           12-12-2009     William E. Lamie         Modified comment(s), added    */
/*                                           resulting in version 5.3    */
/*                                           function prototypes,        */
/*                                           optimized event macro, and  */
/*                                           trace has universal format, */
/*                                           and corrected priority      */
/*                                           assignment in event trace, */
/*                                           changed types to ensure the */
/*                                           new event definitions,      */
/*                                           they are defined elsewhere, */
/*                                           altered preemption- */
/*                                           threshold along with its */
/*                                           priority, resulting in      */
/*                                           version 5.4 */
/*                                           */
#endif

/***************************************************************************/
/***************************************************************************/
/***************************************************************************/
/**                                                                       */
/**   Trace                                                               */
/** ThreadX Component                                                     */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/**                                                                       */
/*******************************************************************************************/

San Diego, CA  92127
11423 West Bernardo Court               http://www.expresslogic.com
Express Logic, Inc.                     info@expresslogic.com
William E. Lamie, Express Logic, Inc.

William E. Lamie, Express Logic, Inc.

5.4

AUTHOR

William E. Lamie, Express Logic, Inc.

DESCRIPTION

This file defines the ThreadX trace component, including constants
and structure definitions as well as external references. It is
assumed that tx_api.h and tx_port.h have already been included.

RELEASE HISTORY

DATE NAME DESCRIPTION

12-12-2009 William E. Lamie Initial Version 5.0
04-09-2007 William E. Lamie Modified comment(s),
resulting in version 5.1
12-12-2008 William E. Lamie Modified comment(s), added
new event definitions,
changed types to ensure the
trace has universal format,
optimized event macro, and
added function logic and new
function prototypes,
resulting in version 5.2
07-04-2009 William E. Lamie Modified comment(s), removed
Filex & Netx event IDs since
they are defined elsewhere,
and corrected priority
assignment in event trace,
resulting in version 5.3
12-12-2009 William E. Lamie Modified comment(s), added
defines for default source,
and added logic to insert
the thread's preemption-
threshold along with its
priority, resulting in
version 5.4

COPYRIGHT (C) 1996-2009 BY EXPRESS LOGIC INC.
THIS SOFTWARE IS COPRIGHTED BY AND IS THE SOLE PROPERTY OF EXPRESS
LOGIC, INC. ALL RIGHTS, TITLE, OWNERSHIP, OR OTHER INTERESTS
IN THE SOFTWARE REMAIN THE PROPERTY OF EXPRESS LOGIC, INC. THIS
SOFTWARE MAY ONLY BE USED IN ACCORDANCE WITH THE CORRESPONDING
LICENSE AGREEMENT. ANY UNAUTHORIZED USE, DUPLICATION, TRANSMISSION,
DISTRIBUTION, OR DISCLOSURE OF THIS SOFTWARE IS EXPRESSLY FORBIDDEN.
THIS COPYRIGHT NOTICE MAY NOT BE REMOVED OR MODIFIED WITHOUT PRIOR
WRITTEN CONSENT OF EXPRESS LOGIC, INC.
*/
/* Event tracing is enabled. */
/* Ensure that the thread component information is included. */
#include "tx_thread.h"

/* Define trace port-specific extension to white space if it isn't defined already. */
#ifndef TX_TRACE_PORT_EXTENSION
#define TX_TRACE_PORT_EXTENSION
#endif

/* Define the default clock source for trace event entry time stamp. The following two item are port specific. 
For example, if the time source is at the address 0x0a800024 and is 16-bits in size, the clock source constants would be: 

#define TX_TRACE_TIME_SOURCE  *(ULONG *) 0x0a800024 
#define TX_TRACE_TIME_MASK  0x0000FFFFUL 
*/
#ifndef TX_TRACE_TIME_SOURCE
#define TX_TRACE_TIME_SOURCE  ++_tx_trace_simulated_time 
#endif
#ifndef TX_TRACE_TIME_MASK
#define TX_TRACE_TIME_MASK  0xFFFFFFFFUL 
#endif

/* Define the ID showing the event trace buffer is valid. */
#define TX_TRACE_VALID  0x54585442UL

/* ThreadX Trace Description. The ThreadX Trace feature is designed to capture events in real-time in a circular event buffer. This buffer may be analysed by other tools. The high-level format of the Trace structure is: 

[Trace Control Header ]
[Trace Object Registry - Entry 0 ]
... 
[Trace Object Registry - Entry "n" ]
[Trace Buffer - Entry 0 ]
... 
[Trace Buffer - Entry "n" ]
*/

/* Trace Control Header. The Trace Control Header contains information that defines the format of the Trace Object Registry as well as the location and current entry of the Trace Buffer itself. The high-level format of the Trace Control Header is: 

Entry Size Description 
[Trace ID] 4 This 4-byte field contains the ThreadX Trace Identification. If the trace buffer is valid, the contents are 0x54585442 (TEXT). Since it is written as a 32-bit unsigned word, this value is also used to determine if the event trace information is in little or big endian format. 
[Timer Valid Mask] 4 Mask of valid bits in the 32-bit time stamp. This enables use of 32, 24, 16, or event 8-bit timers. If the time source is 32-bits, the mask is 0xFFFFFFFF. If the time source is 16-bits, the mask is 0x0000FFFF. 
[Trace Base Address] 4 The base address for all trace pointer. Subtracting the pointer and this address will yield the proper offset into the trace buffer. 
[Trace Object Registry Start Pointer] 4 Pointer to the start of Trace Object Registry 
[Trace Object Registry Size] 2 Reserved two bytes - should be 0x0000 
[Trace Object Registry Name Size] 4 Number of bytes in object name 
[Trace Buffer End Pointer] 4 Pointer to the end of Trace Buffer Area 
[Trace Buffer Current Pointer] 4 Pointer to the oldest entry in the Trace Buffer. This entry will be overwritten on the next event and incremented to the next event (wrapping to the top if the buffer end pointer is exceeded). 
*/

/* Define the Trace Control Header. */
typedef struct TX_TRACE_CONTROL_HEADER_STRUCT 
{
    ULONG tx_trace_control_header_id; 
    ULONG tx_trace_control_header_timer_valid_mask; 
    ULONG tx_trace_control_header_trace_base_address; 
    ULONG tx_trace_control_header_object_registry_start_pointer; 
    USHORT tx_trace_control_header_object_registry_size; 
    USHORT tx_trace_control_header_object_registry_name_size; 
    ULONG tx_trace_control_header_trace_object_registry_current_pointer; 
} TX_TRACE_CONTROL_HEADER_STRUCT;
**ThreadX Trace Events and Constants**

- **(ThreadX Trace Thread Suspend)**
  ```c
  #define TX_TRACE_THREAD_SUSPEND 2
  /* I1 = thread ptr, I2 = new_state, I3 = stack ptr, I4 = next thread */
  ```

- **(ThreadX Trace Thread Resume)**
  ```c
  #define TX_TRACE_THREAD_RESUME 1
  /* I1 = thread ptr, I2 = previous_state, I3 = stack ptr, I4 = next thread */
  ```

User-defined event numbers start at 4096 and continue through 65535, as defined by the constants `TX_TRACE_USER_EVENT_START` and `TX_TRACE_USER_EVENT_END`, respectively. User events should be based on these constants in case the user event number assignment is changed in future releases.

**Event Numbers**
- **USBX events**: 600-999
- **NetX events**: 300-599
- **FileX events**: 200-299
- **ThreadX events**: 1-199

Event numbers 0 through 4095 are reserved by Express Logic. Specific event assignments are:

**Trace Buffer Entry**

The Trace Buffer Entry contains information about a particular event in the system. The high-level format of the Trace Buffer Entry is:

```c
typedef struct TX_TRACE_OBJECT_REGISTRY_ENTRY_STRUCT

UCHAR tx_trace_object_registry_entry_object_type; /* Types defined above */
UCHAR tx_trace_object_registry_entry_object_parameter_1; /* Parameter value defined */
UCHAR tx_trace_object_registry_entry_object_parameter_2; /*   according to type above */
USHORT tx_trace_object_registry_entry_object_available; /* TX_TRUE -> available */
USHORT tx_trace_object_registry_entry_object_available; /* Should be zero - except for thread */
USHORT tx_trace_object_registry_entry_object_reserved1; /* Should be zero - except for thread */
USHORT tx_trace_object_registry_entry_object_reserved2; /* Should be zero - except for thread */
USHORT tx_trace_object_registry_entry_object_registry_end_pointer; /* Should be zero - except for thread */
USHORT tx_trace_object_registry_entry_object_registry_end_pointer; /* Should be zero - except for thread */
USHORT tx_trace_object_registry_entry_object_name[TX_TRACE_OBJECT_REGISTRY_NAME]; /* Should be zero - except for thread */
USHORT tx_trace_object_registry_entry_object_name[TX_TRACE_OBJECT_REGISTRY_NAME]; /* Should be zero - except for thread */
USHORT tx_trace_object_registry_entry_object_name[TX_TRACE_OBJECT_REGISTRY_NAME]; /* Should be zero - except for thread */
```
#define tx_trace_buffer_entry_thread_priority
ULONG tx_trace_buffer_entry_thread_priority;

#define TX_TRACE_TIMER_PERFORMANCE_INFO_GET 128 /* I1 = timer ptr */
#define TX_TRACE_TIMER_INFO_GET 127 /* I1 = timer ptr, I2 = stack ptr */
#define TX_TRACE_TIMER_DELETE 126 /* I1 = timer ptr */
#define TX_TRACE_TIMER_DEACTIVATE 125 /* I1 = timer ptr, I2 = stack ptr */
#define TX_TRACE_TIMER_ACTIVATE 122 /* I1 = timer ptr */
#define TX_TRACE_TIME_SET 121 /* I1 = new time */
#define TX_TRACE_TIME_GET 120 /* I1 = current time, I2 = stack ptr */
#define TX_TRACE_THREAD_STACK_ERROR_NOTIFY 113 /* None */
#define TX_TRACE_THREAD_SLEEP 112 /* I1 = sleep value, I2 = thread state, I3 = stack ptr */
#define TX_TRACE_THREAD_RESET 110 /* I1 = thread ptr, I2 = thread state */
#define TX_TRACE_THREAD_PREEMPTION_CHANGE 107 /* I1 = thread ptr, I2 = new threshold, I3 = old threshold, I4 = thread state */
#define TX_TRACE_THREAD_PRIORITY_CHANGE 106 /* I1 = thread ptr, I2 = new priority, I3 = old priority, I4 = thread state */
#define TX_TRACE_THREAD_RELQUEUISH 105 /* I1 = stack ptr, I2 = thread */
#define TX_TRACE_THREAD_RESUME_API 104 /* I1 = thread ptr, I2 = thread state, I3 = stack ptr */
#define TX_TRACE_THREAD_SLEEP 103 /* I1 = sleep value, I2 = thread state, I3 = stack ptr */
#define TX_TRACE_THREAD_STACK_ERROR_NOTIFY 102 /* I1 = semaphore ptr, I2 = current count, I3 = suspended count */
#define TX_TRACE_SEMAPHORE_CEILING_PUT 80 /* I1 = semaphore ptr, I2 = current count, I3 = suspended count, I4 = ceiling */
#define TX_TRACE_QUEUE_PRIORITIZE 67 /* I1 = queue ptr, I2 = suspended count, I3 = stack ptr */
#define TX_TRACE_MUTEX_PUT 57 /* I1 = mutex ptr, I2 = owning thread, I3 = own count, I4 = stack ptr */
#define TX_TRACE_MUTEX_PRIORITIZE 56 /* I1 = mutex ptr, I2 = suspended count, I3 = stack ptr */
#define TX_TRACE_MUTEX_CREATE 50 /* I1 = mutex ptr, I2 = inheritance, I3 = stack ptr */
#define TX_TRACE_INTERRUPT_CONTROL 40 /* I1 = new interrupt posture, I2 = stack ptr */
#define TX_TRACE_EVENT_FLAGS_SET_NOTIFY 37 /* I1 = group ptr */
#define TX_TRACE_EVENT_FLAGS_PERFORMANCE_SYSTEM_INFO_GET 35 /* None */
#define TX_TRACE_EVENT_FLAGS_INFO_GET 33 /* I1 = group ptr */
#define TX_TRACE_EVENT_FLAGS_CREATE 30 /* I1 = group ptr, I2 = stack ptr */
#define TX_TRACE_BYTE_POOL_INFO_GET 23 /* I1 = pool ptr */
#define TX_TRACE_BYTE_POOL_DELETE 22 /* I1 = pool ptr */
#define TX_TRACE_BYTE_POOL_CREATE 21 /* I1 = pool ptr, I2 = stack ptr */
#define TX_TRACE_BYTE_POOL_INFO_SET 20 /* I1 = pool ptr */
#define TX_TRACE_BYTE_POOL_PRIORITY_INFO_GET 19 /* None */
#define TX_TRACE_BYTE_POOL_PRIORITY_INFO_SET 18 /* None */
#define TX_TRACE_BLOCK_POOL_PERFORMANCE_SYSTEM_INFO_GET 14 /* I1 = pool ptr */
#define TX_TRACE_BLOCK_POOL_INFO_GET 13 /* I1 = pool ptr */
#define TX_TRACE_BLOCK_POOL_CREATE 12 /* I1 = pool ptr, I2 = stack ptr */
#define TX_TRACE_BLOCK_POOL_DELETE 11 /* I1 = pool ptr, I2 = memory ptr, I3 = suspended, I4 = available bytes */
#define TX_TRACE_BLOCK_POOL_PRIORITY 10 /* I1 = pool ptr, I2 = memory ptr, I3 = suspended, I4 = available bytes */
#define TX_TRACE_BLOCK_ALLOCATE 9 /* I1 = pool ptr, I2 = memory ptr, I3 = suspended, I4 = available bytes */
#define TX_TRACE_BLOCK_FREE 8 /* I1 = pool ptr, I2 = memory ptr, I3 = suspended, I4 = available bytes */
#define TX_TRACE_Block_Free_2 7 /* I1 = pool ptr, I2 = memory ptr, I3 = suspended, I4 = available bytes */
#define TX_TRACE_BLOCK_FREE_2 6 /* I1 = pool ptr, I2 = memory ptr, I3 = suspended, I4 = available bytes */
#define TX_TRACE_EVENT_FLAG 5 /* I1 = new thread ptr, I2 = system state, I3 = preemption disable, I4 = stack ptr */
#define TX_TRACE_ISR_EXIT 4 /* I1 = stack_ptr, I2 = ISR number, I3 = system state, I4 = preemption disable */
#define TX_TRACE_TIME_SLICE 3 /* I1 = new thread ptr, I2 = system state */
#define TX_TRACE_RUNNING 2 /* None */
#define TX_TRACE_TRACE 1 /* None */

/* Define the rest of the ThreadX system events. */

/* Define the rest of the ThreadX system events. */

typedef struct tx_trace_buffer_entry_struct |
ULONGLONG tx_trace_buffer_entry_thread_pointer;
ULONGLONG tx_trace_buffer_entry_thread_priority;
ULONGLONG tx_trace_buffer_entry_event_id;
ULONG tx_trace_buffer_entry_time_stamp;
ULONG tx_trace_buffer_entry_information_field_1;
ULONG tx_trace_buffer_entry_information_field_2;
ULONG tx_trace_buffer_entry_information_field_3;
ULONG tx_trace_buffer_entry_information_field_4;
}

TX_TRACE_BUFFER_ENTRY;

/* Trace management component data declarations follow. */
/* Determine if the initialization function of this component is including
this file. If so, make the data definitions really happen. Otherwise,
make them extern so other functions in the component can access them. */
#ifdef TX_TRACE_INIT
#define TRACE_DECLARE extern
#else
#define TRACE_DECLARE
#endif

/* Define the pointer to the start of the trace buffer control structure. */
TRACE_DECLARE TX_TRACE_CONTROL_HEADER *_tx_trace_control_header_ptr;

/* Define the pointer to the start of the trace object registry area in the trace buffer. */
TRACE_DECLARE TX_TRACE_OBJECT_REGISTRY_ENTRY *_tx_trace_registry_start_ptr;

/* Define the pointer to the end of the trace object registry area in the trace buffer. */
TRACE_DECLARE TX_TRACE_OBJECT_REGISTRY_ENTRY *_tx_trace_registry_end_ptr;

/* Define the pointer to the starting entry of the actual trace event area of the trace buffer. */
TRACE_DECLARE TX_TRACE_BUFFER_ENTRY * _tx_trace_buffer_start_ptr;

/* Define the pointer to the ending entry of the actual trace event area of the trace buffer. */
TRACE_DECLARE TX_TRACE_BUFFER_ENTRY * _tx_trace_buffer_end_ptr;

/* Define the pointer to the current entry of the actual trace event area of the trace buffer. */
TRACE_DECLARE TX_TRACE_BUFFER_ENTRY * _tx_trace_buffer_current_ptr;

/* Define the trace event enable bits, where each bit represents a type of event that can be enabled
or disabled dynamically by the application. */
TRACE_DECLARE ULONG _tx_trace_event_enable_bits;

/* Define a counter that is used in environments that don’t have a timer source. This counter
is incremented on each use giving each event a unique timestamp. */
TRACE_DECLARE ULONG _tx_trace_simulated_time;

/* Define the function pointer used to call the application when the trace buffer wraps. If NULL,
the application has not registered a callback function. */
TRACE_DECLARE VOID (*_tx_trace_full_notify_function)(VOID *);

/* Define the event trace macros that are expanded in-line when event tracing is enabled. */
#define TX_TRACE_INITIALIZE _tx_trace_initialize();
#define TX_TRACE_OBJECT_REGISTER(t,p,n,a,b) _tx_trace_object_register(t, (VOID *) p, (CHAR *) n, (ULONG) a, (ULONG) b);
#define TX_TRACE_OBJECT_UNREGISTER(o) _tx_trace_object_unregister((VOID *) o);
#define TX_TRACE_IN_LINE_INSERT(i,a,b,c,d,e) \
{ 
TX_TRACE_BUFFER_ENTRY *trace_event_ptr; 
ULONG trace_system_state; 
ULONG trace_priority; 
TX_THREAD *trace_thread_ptr; 
trace_event_ptr = _tx_trace_buffer_current_ptr; 
if ((trace_event_ptr) && (_tx_trace_event_enable_bits & ((ULONG) (e)))) \
{ 
TX_TRACE_PORT_EXTENSION
trace_system_state = (ULONG) _tx_thread_system_state; 
trace_thread_ptr = _tx_thread_current_ptr; 
if ((trace_system_state == 0) \
{ 
trace_priority = _tx_thread_priority; 
else if (trace_system_state < 0x80808080) \
{ 
trace_priority = (ULONG) trace_thread_priority; 
trace_thread_ptr = (TX_THREAD *) 0xFFFFFFFFUL; 
}
else \
{
    trace_thread_ptr = (TX_THREAD *) 0xF0F0F0F0UL; \n    trace_priority = 0; \n} \
\} \
trace_event_ptr -> tx_trace_buffer_entry_thread_pointer = (ULONG) trace_thread_ptr; \ntrace_event_ptr -> tx_trace_buffer_entry_thread_priority = (ULONG) trace_priority; \ntrace_event_ptr -> tx_trace_buffer_entry_event_id = (ULONG) (i); \ntrace_event_ptr -> tx_trace_buffer_entry_time_stamp = (ULONG) TX_TRACE_TIME_SOURCE; \ntrace_event_ptr -> tx_trace_buffer_entry_information_field_1 = (ULONG) (a); \ntrace_event_ptr -> tx_trace_buffer_entry_information_field_2 = (ULONG) (b); \ntrace_event_ptr -> tx_trace_buffer_entry_information_field_3 = (ULONG) (c); \ntrace_event_ptr -> tx_trace_buffer_entry_information_field_4 = (ULONG) (d); \ntrace_event_ptr++; \
if (trace_event_ptr >= _tx_trace_buffer_end_ptr) \
{ 
    trace_event_ptr = _tx_trace_buffer_start_ptr; \n    _tx_trace_buffer_current_ptr = trace_event_ptr; \n    _tx_trace_control_header_ptr -> tx_trace_control_header_buffer_current_pointer = (ULONG) trace_event_ptr; \n    if (_tx_trace_full_notify_function) (VOID *) _tx_trace_control_header_ptr; \n} \
else \
{ 
    _tx_trace_buffer_current_ptr = trace_event_ptr; \n    _tx_trace_control_header_ptr -> tx_trace_control_header_buffer_current_pointer = (ULONG) trace_event_ptr; \n} \
\}
#endif

/* Define function prototypes of the trace component. */
UINT    _tx_trace_enable(VOID *trace_buffer_start, ULONG trace_buffer_size, ULONG registry_entries);
UINT    _tx_trace_event_filter(ULONG event_filter_bits);
UINT    _tx_trace_event_unfilter(ULONG event_filter_bits);
UINT    _tx_trace_disable(VOID);
VOID    _tx_trace_initialize(VOID);
UINT    _tx_trace_interrupt_control(UINT new_posture);
VOID    _tx_trace_isr_enter_insert(ULONG isr_id);
VOID    _tx_trace_isr_exit_insert(ULONG isr_id);
VOID    _tx_trace_object_register(UCHAR object_type, VOID *object_ptr, CHAR *object_name, ULONG parameter_1, ULONG parameter_2);
VOID    _tx_trace_object_unregister(VOID *object_ptr);
UINT    _tx_trace_buffer_full_notify(VOID (*full_buffer_callback)(VOID *));
UINT    _tx_trace_user_event_insert(ULONG event_id, ULONG info_field_1, ULONG info_field_2, ULONG info_field_3, ULONG info_field_4);
#endif
There are three DOS command line utilities found in the TraceX installation under the *Utilities* sub-directory. They are described on the following page.

- ea2tracex.exe
- hex2tracex.exe
- mot2tracex.exe
The utilities supplied are listed below:

<table>
<thead>
<tr>
<th>Utility</th>
<th>Purpose</th>
<th>Command Line Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ea2tracex.exe</td>
<td>Converts the trace file generated by ThreadX in association with the GHS tools to the TraceX trace file format. Note: ThreadX for GHS tools produces a different trace format than ThreadX for non-GHS tools, which is why this conversion utility is needed.</td>
<td><code>&gt; ea2tracex original_file converted_file &lt;cr&gt;</code></td>
</tr>
<tr>
<td>hex2tracex.exe</td>
<td>Converts a trace file generated by ThreadX but dumped from the development tool in Intel HEX format to the binary TraceX trace file format. Note that TraceX V5 and above can open HEX files without converting them.</td>
<td><code>&gt; hex2tracex hex_file converted_file &lt;cr&gt;</code></td>
</tr>
<tr>
<td>mot2tracex.exe</td>
<td>Converts a trace file generated by ThreadX but dumped from the development tool in Motorola S-Record format to the binary TraceX trace file format. Note that TraceX V5 and above can open S-Record files without converting them.</td>
<td><code>&gt; mot2tracex mot_file converted_file &lt;cr&gt;</code></td>
</tr>
</tbody>
</table>
Dumping the trace buffer created by ThreadX to a file on the host computer is done via commands and/or utilities provided by the specific development tool being used. This appendix contains examples of dumping a trace buffer to a host file in some of the more popular development tools used with ThreadX.

- BenchX Tools 258
- RealView Tools 259
- IAR Tools 259
- CodeWarrior Tools 260
- MPLAB Tools 261
- GHS Tools 267
- Renesas HEW 268
BenchX Tools

The trace buffer can be dumped to a host file easily with the BenchX tools by selecting the *Store Memory To File* button within the *Memory View*, as shown below:

At this point, specify the trace buffer address, size, destination file name (including path), and select the *Save* button as shown below. This will create the binary trace file for viewing within TraceX.
RealView Tools

The trace buffer can be dumped to a host file easily with the ARM RealView tools by entering the following command at the command line prompt in RealView:

```
> WRITEFILE,raw trace_file.trx=0x6860..0xE560
```

Upon completion, the file `trace_file.trx` will contain the trace buffer that is located starting at the address 0x6860 and goes up to address 0xE560. This file is ready for viewing by TraceX.

IAR Tools

The trace buffer can be dumped to a host file very easily with the IAR tools by simply right clicking in the memory view and selecting the **Memory Save**... option, as shown below.
This results in the **Memory Save** dialog to be displayed. Enter the starting and ending address and the trace file name, then select the **Save** button. In the example shown below, the IAR tools save the specified trace buffer into Intel HEX records in the file `trace_file.hex`.

![Memory Save Dialog](image)

At this point, we have the trace buffer saved in the `trace_file.hex` file on the host and is ready for viewing with TraceX.

**CodeWarrior Tools**

The trace buffer can be dumped to a host file easily with the CodeWarrior tools by entering the `save` command in the Command Window. The following example `save` command assumes the trace buffer starts at 0x102200 and ends at 0x109F00:

```
> save -b p:0x102200..0x109F00 trace_file.trx -a 32bit
```
This results in the trace buffer being saved in the file `trace_file.trx` on the host.

**MPLAB Tools**

MPLAB can create a TraceX-compatible trace file through its Export Table utility, which allows the export of any range of memory to a host file. To use this utility to create a trace file for TraceX, proceed as follows:

**Step 1**

Open a memory window by selecting View -> Memory.
Step 2

Right-click within the Memory View to display a list of options. Specify Display Format -> 1 Byte to select byte display.
Step 3

Right-click again within the Memory View Window and select Go To, which opens a dialog box that enables you to specify the address of the event buffer. This example shows event_buffer being displayed.
Step 4

This highlights the contents of the first location of the trace buffer, which is always the string BTXT....
Step 5  
Now, right-click again to bring up the options menu, and select Export Table.

![Image of Export Table dialog]

Step 6  
This brings up the Export Table dialog, as shown. Specify the range of addresses to export. For an 8K trace buffer, as is the case in this example, specify the range 0xA00006AC to 0xA00026AC, and enter a
name for the host file to be created (demo_threadx.trx in this example).

Step 7

A file named demo_threadx.trx will be created on the host, and this file can be opened by TraceX.

GHS Tools

The trace buffer can be dumped to a host file easily with the GHS tools by entering the following command at the command line prompt in the debug command window:

```
memdump raw c:\release\threadx\demo_threadx.trx event_buffer 32768
```

Upon completion, the file demo_threadx.trx will contain the trace buffer that is located in the event_buffer with a size of 32,768 bytes and is ready for viewing by TraceX.
The trace buffer can be dumped to a host file easily with the Renesas HEW tools by following the three steps (and sub-steps) below:

**Step 1**
Open Memory Window.
Step 2
Place cursor within memory window and right click.

Step 3
Select Save, then in the Save Memory As window do the following:

- Select File format: Binary.
- Specify Filename: As Desired
- Specify Start address: trace_buffer
- Specify End address: (trace_buffer+size)
- Specify Access size: 1
• Click Save
Symbols
.NET framework installation 23
.NET v3.5 21

A
actual time display 50

B
base address of all trace file pointers 66
BenchX tools
dumping trace buffer 258
buffer start and end pointers 234
building an event buffer 69
byte order of file 66

C
CodeWarrior tools
dumping trace buffer 260
completing TraceX installation 21
context
Interrupt 37
moving 38
Relative Time 43
System Timer Thread 37
control header ID 233
controlling elements of TraceX display 30
current event display 44
Customer Support Center information 9

D
default location for TraceX installation 14
default view mode 33
defining time-stamp constants 70
delta number of ticks between events 49
demo_threadx.trx 25, 43
deterministic condition 51
deterministic priority inversion 52
deterministic priority inversion range 51
disable event tracing 84
display
zoom in 47
display mode tabs 32
displaying all events on same line 35
displaying system events 32
DOS command line utilities 255
dumping the trace buffer 257

E
ea2trace.exe 256
enable event tracing 74
enabling event trace 70
event ID 239
event information
detailed 42
event information display 42
event searching 46
event trace buffer current pointer 235
event trace entries 238
event trace object registry 235
event trace support 70
example of time-stamp definition 71
example trace files 25
execution profile 56
exporting the trace buffer 72
extended event trace API 73
F
FileX
   performance statistics 63
   statistics 56
FileX events 117, 118
format of event trace buffer 74, 76, 80, 231
format of the ThreadX event trace buffer 232
format of tx_trace_disable 84
format of tx_trace_isr_entry_insert 86
format of tx_trace_isr_exit_insert 88
format of tx_trace_user_event_insert 92
_fx_version_id 10

G
general flow of .NET installation 22
GHS tools
   dumping trace buffer 267
Guide 9
guide cConventions 9

H
hex2tracex.exe 256
high frequency timer 35

I
IAR tools
   dumping trace buffer 259
information fields 239
Initialize/Idle 37
insert ISR entry event 86
insert ISR exit event 88
insert user event 92
installation dialogs 15
Intel HEX 72

L
latest product information 9
launching the TraceX 16
launching TraceX 24
layout of the event trace buffer 232

M
making support requests 10
maximum number of bytes for each object name 66
memory area dump 72
Microsoft .NET 14
Microsoft .NET framework 12
mot2tracex.exe 256
Motorola S-Record file format 72
MPlab tools
   dumping trace buffer 261
multiple event viewing 44

N
NetX
   performance statistics 65
   statistics 56
NetX events 135, 136
non-deterministic priority inversion 52
non-deterministic priority inversions 51
number of ticks between events 49
_nx_version_id 10

O
object available flag 235
object entry type 236
object name 237
object parameters 237
object pointer 237
object registry entry 235
object types 236
opened trace file information 66
overview of system activity 34

P
performance analysis 56
performance statistics 56, 61
popular services 56, 58
post-mortem tool 70
priority inversions 51, 62
R
Raw Time Stamp  43
raw trace dump  67
RealView tools
   dumping trace buffer  259
register trace buffer full application callback  90
registry name size  234
registry start and end pointer points  234
relative event numbers  33
relative ticks from beginning of trace  35
restoring to full icon view  48
run-time event
   information  42

S
search parameters
   primary  46
sending comments to Express Logic  10
Sequential View  33
sequential view mode  33
Setup.exe  14
single summary line  35
size of time source  66
standard information  43
summary of context  36
support email  9
support engineer contact information  9
system contexts  36
system performance tuning  34

T
terms of the license agreement  16
thread pointer  238
thread priority  239
thread stack usage  56
thread state changes  70
ThreadX  7,  12
   stack usage  60
ThreadX events  95,  96
ThreadX event-trace capability  70
ThreadX object types  237
ticks between events  49
time relative  34
Time Stamp  43
time stamp  35
Time View  33
time view mode  34
timer valid mask  233
trace buffer base address  234
trace buffer in a binary  72
trace event entry  238
trace_file.hex  260
trace_file.trx  259,  261
TraceFiles subdirectory  25
TraceX
   building an event buffer  69
definition  11
   installation  13
   main display window  28
   overall functionality  27
   system event display  32
   title bar  29
tool bar  30
tool bar buttons  30
   using  24
   version  29
TraceX constraints  12
TraceX execution profile  57
TraceX files  12
TraceX graphic user interface  24
TraceX performance analysis  55
   _tx_build_options  10
   TX_ENABLE_EVENT_TRACE  70,  73,  231
tx_port.h  70,  233,  241
tx_thread_create  234
tx_trace.h  239,  247
tx_trace_enable  70,  72,  84,  231,  234,  235,  238
   TX_TRACE_OBJECT_REGISTRY_NAME  234
   TX_TRACE_TIME_MASK  233
   TX_TRACE_USER_EVENT_START  92
   _tx_version_id  10
U
unfilter specified events  80
user responsibility  11
user-defined events  42,  70
Utilities sub-directory  255

V
view of location of the system processing
34

Z
zoom out  47
zooming  47
TraceX® User’s Manual: Software