Contents

LevelX

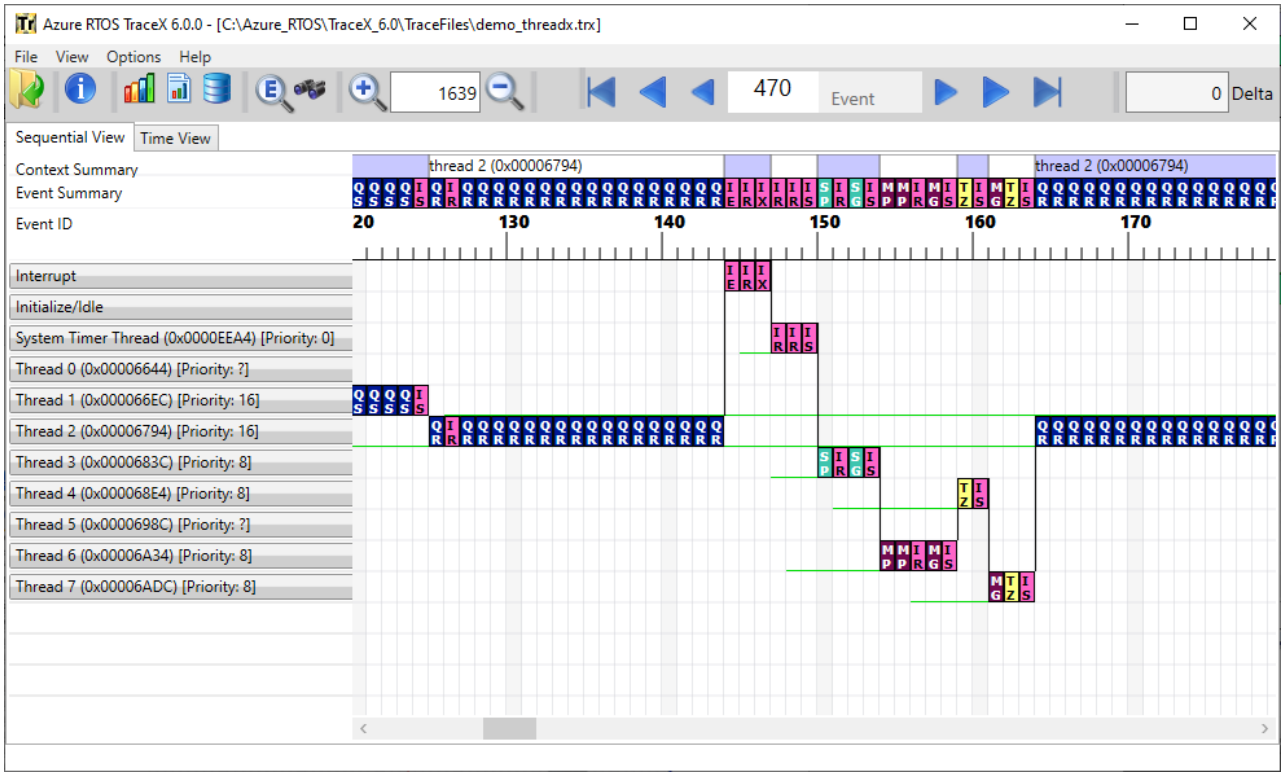
USBX

TraceX

Overview of Azure RTOS TraceX

7/20/2020 • 9 minutes to read

Azure RTOS TraceX is Microsoft's host-based analysis tool that provides developers with a graphical view of real-time system events and enables them to visualize and better understand the behavior of their real-time systems. With Azure RTOS TraceX, developers can see clearly the occurrence of system events like interrupts and context switches that occur out of view of standard debugging tools. The ability to identify and study these events, and to pinpoint the timing of their occurrence in the context of the overall system's operation enables developers to resolve programming problems by finding unexpected behavior and letting them investigate specific areas further Trace information is stored in a buffer on the target system, with the buffer location and size determined by the application at run-time. Azure RTOS TraceX can process any buffer constructed in the proper manner, not only from Azure RTOS ThreadX, but from any application or RTOS. The trace information may be uploaded to the host for analysis at any time – either post mortem or upon a breakpoint. Azure RTOS ThreadX implements a circular buffer, which enables the most recent "N" events to be available for inspection in the event of system malfunction or other significant event.



TraceX Single-Core Display

Key capabilites

Azure RTOS TraceX built-in system analysis

Azure RTOS TraceX is provides built-in system analysis reports that are available via a single button click from the TraceX toolbar. These buttons and reports include:

Generate Execution Profile report

Generate Performance Statistics report

Generate Thread Stack Usage report

Trace data collected By Azure RTOS ThreadX

Azure RTOS TraceX is designed to work with Azure RTOS ThreadX, which constructs a database of system and application "events" on the target system during run-time. These events include:

- thread context switches
- preemptions
- suspensions
- terminations
- system interrupts
- application-specific events
- all Azure RTOS ThreadX API calls
- all Azure RTOS NetX API calls
- all Azure RTOS FileX API calls
- all Azure RTOS USBX API calls
- application-defined icons and information

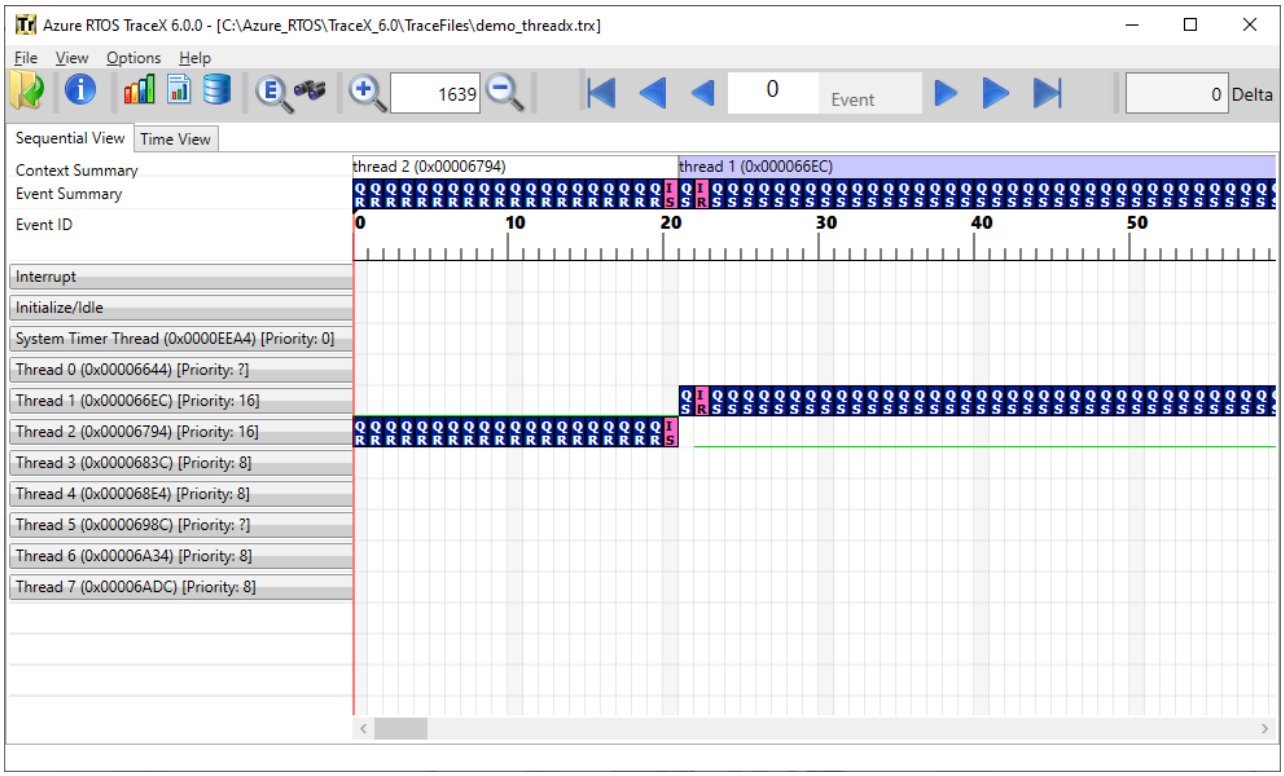
Events are logged under program control, with time-stamping and active thread identification so they can be displayed later in the proper time sequence, and associated with the appropriate thread. Event logging may be stopped and restarted by the application program dynamically, for example, when an area of interest is encountered. This avoids cluttering the database and using up target memory when the system is performing correctly.

Azure RTOS TraceX is like a software logic analyzer

Once the event log has been uploaded from target memory to the host, Azure RTOS TraceX displays the events graphically on a horizontal axis representing time, with the various application threads and system routines to which the events are related listed along the vertical axis. Azure RTOS TraceX creates a "software logic analyzer" on the host, making system events plainly visible. Events are represented by color coded icons, located at the point of occurrence along the horizontal timeline, to the right of the relevant thread or system routine. When an event icon is selected, the corresponding information for that event is displayed, as well as the information for the two previous and two subsequent events. This provides quick, single-click access to the most immediate information about the event and its immediately surrounding events. Azure RTOS TraceX provides a "Summary" display that shows all system events on a single horizontal line to simplify analysis of systems with many threads.

Sequential view mode

The sequential view mode is selected by clicking the "Sequential View" tab. 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. It shows the relative event number from the beginning of the trace. This mode is the default mode and is especially useful in getting a good overview of what is going on in the system.

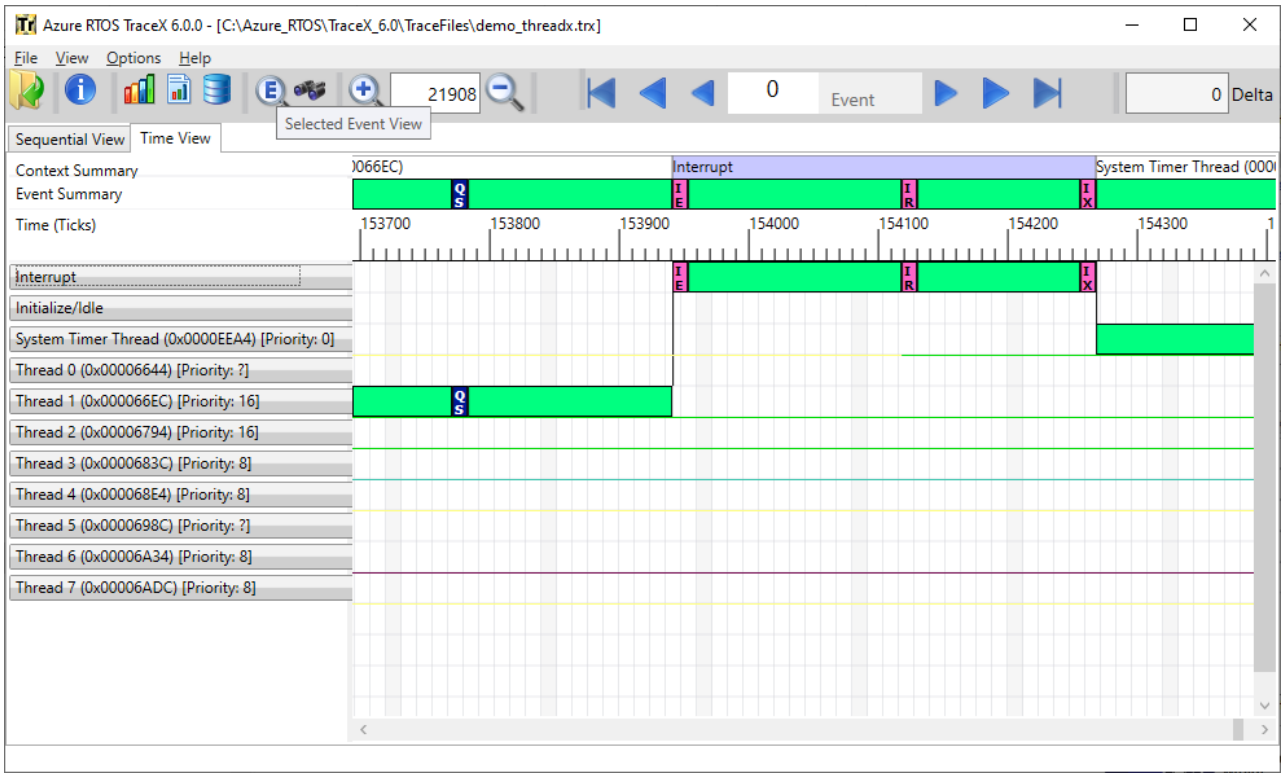


Sequential view mode

Time view mode

In this mode, events are shown in a time relative manner–with a solid green bar being used to show execution between events. This mode is especially 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.

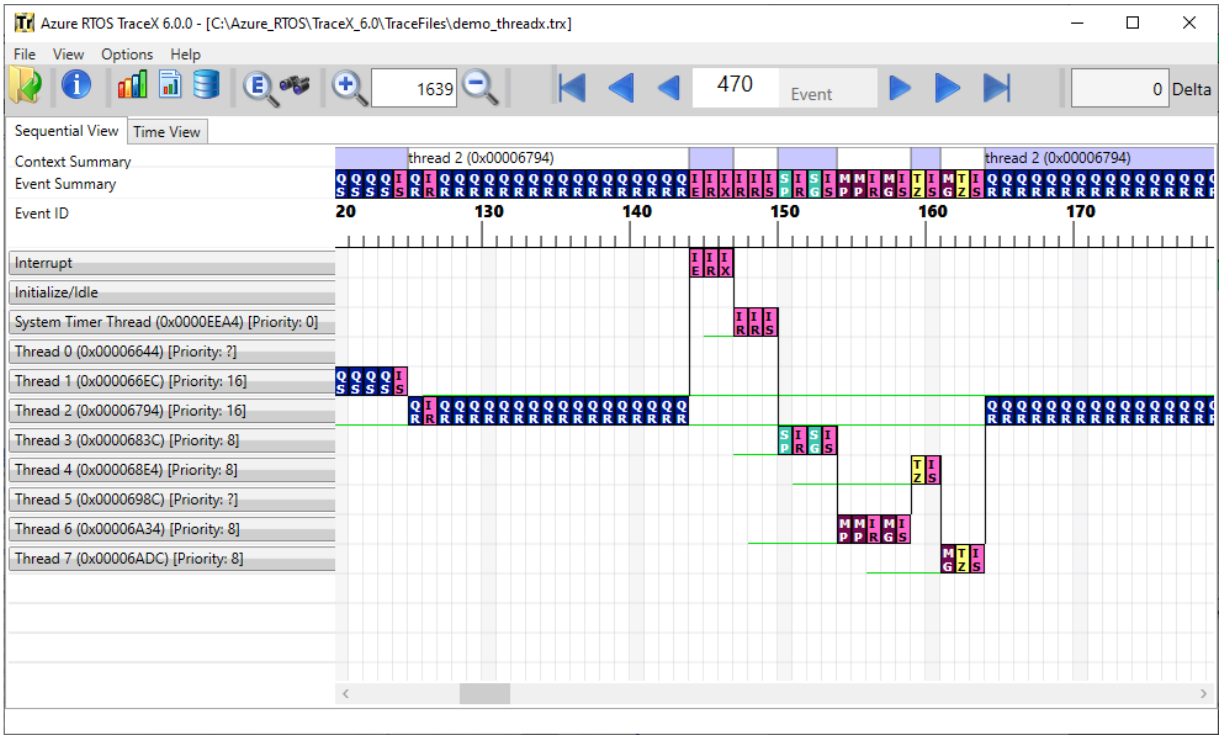
Note also the ruler above the event display. This ruler shows relative ticks from the beginning of the trace, as derived from the time stamp instrumented in the event trace logging inside of Azure RTOS ThreadX. If the time-stamps are too close (low frequency timer), the events will run together. Conversely, 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

Azure RTOS TraceX also provides a single summary line that includes all events on the same line. The summary line contains a summary of the context as well as the corresponding event summary underneath. This makes it easy to see an overview of a complex system. The summary bar is especially beneficial in systems that have a great number of threads. Without such a summary line, the user would have to follow complex system interactions using the vertical scroll bar to follow the context of execution.

Azure RTOS TraceX lists the system contexts on the left-hand side of the display. Events that occur in a particular context are displayed on the horizontal line to the right of that context. In this way, the user 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.

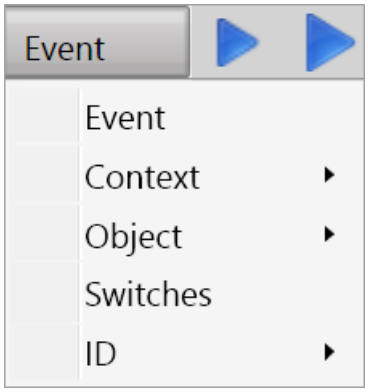


System Summary Line

The first two context entries are always the “Interrupt” and “Initialize/Idle” contexts. The “Interrupt” context represents all system events made from Interrupt Service Routines (ISRs). The “Initialize/Idle” context represents two contexts in Azure RTOS 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.

Methods of navigation

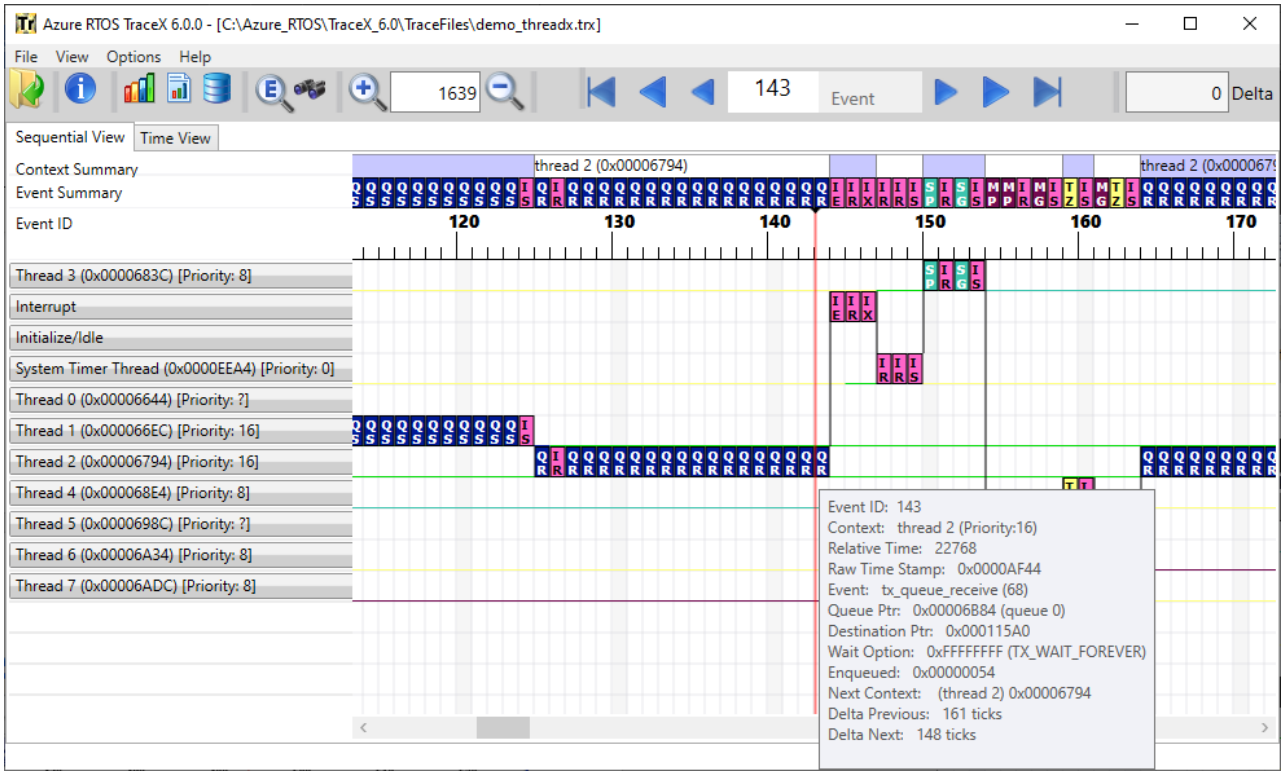
Azure RTOS TraceX enables the developer to specify how the "Next" and "Previous" navigation buttons operate.



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 same context. If “Object” is selected, navigation is done on the next/previous event of the current object, e.g., events associated with a specific queue. If “Switches” is selected, navigation is done on the next/previous context switch. If “Same ID” is selected, navigation is done on the next/previous event for the same event ID.

Event information display

Azure RTOS TraceX provides detailed information on some 300 events. These include six internal Azure RTOS ThreadX events, two ISR events (enter and exit), 14 internal Azure RTOS FileX events, 42 internal Azure RTOS NetX events, and one user-defined event. The remaining events correspond directly to Azure RTOS ThreadX, Azure RTOS FileX, and Azure RTOS NetX API services. 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 here:

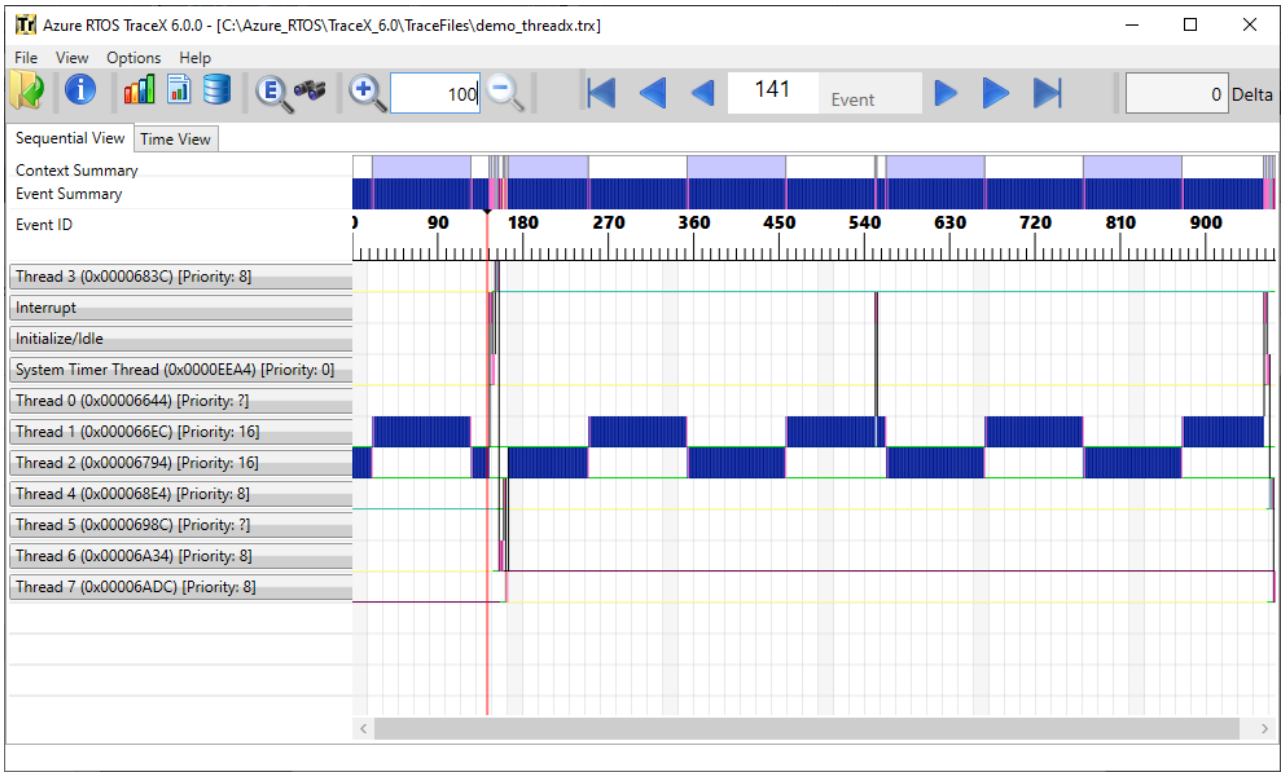


Mouse-Over Displays More Info

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.

Zooming in and out

By default, Azure RTOS TraceX displays the events in an easy-to-view size, with a 1:1 pixel:tick mapping. The user may zoom in or zoom out as desired. Zooming out to 100% is useful to see the entire trace in the current display view, while zooming in is useful in conditions where the events overlap due to the resolution of the time stamp source.



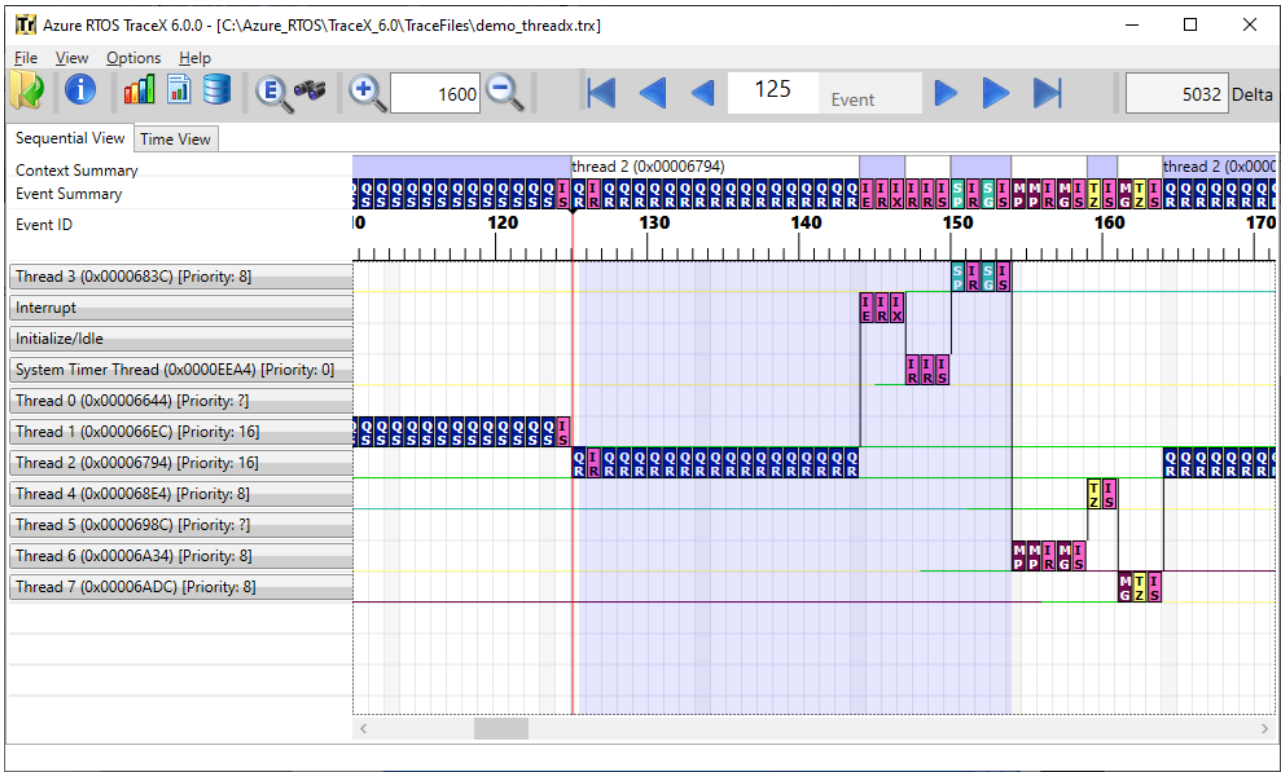
Zoom Out to 100% View or Zoom In for Details

When zoomed out at 100% – showing 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 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 Azure RTOS TraceX is easy–simply click on the starting event and drag the mouse to the ending event. The delta number of ticks between the events will show up in the upper right-hand corner of the display.



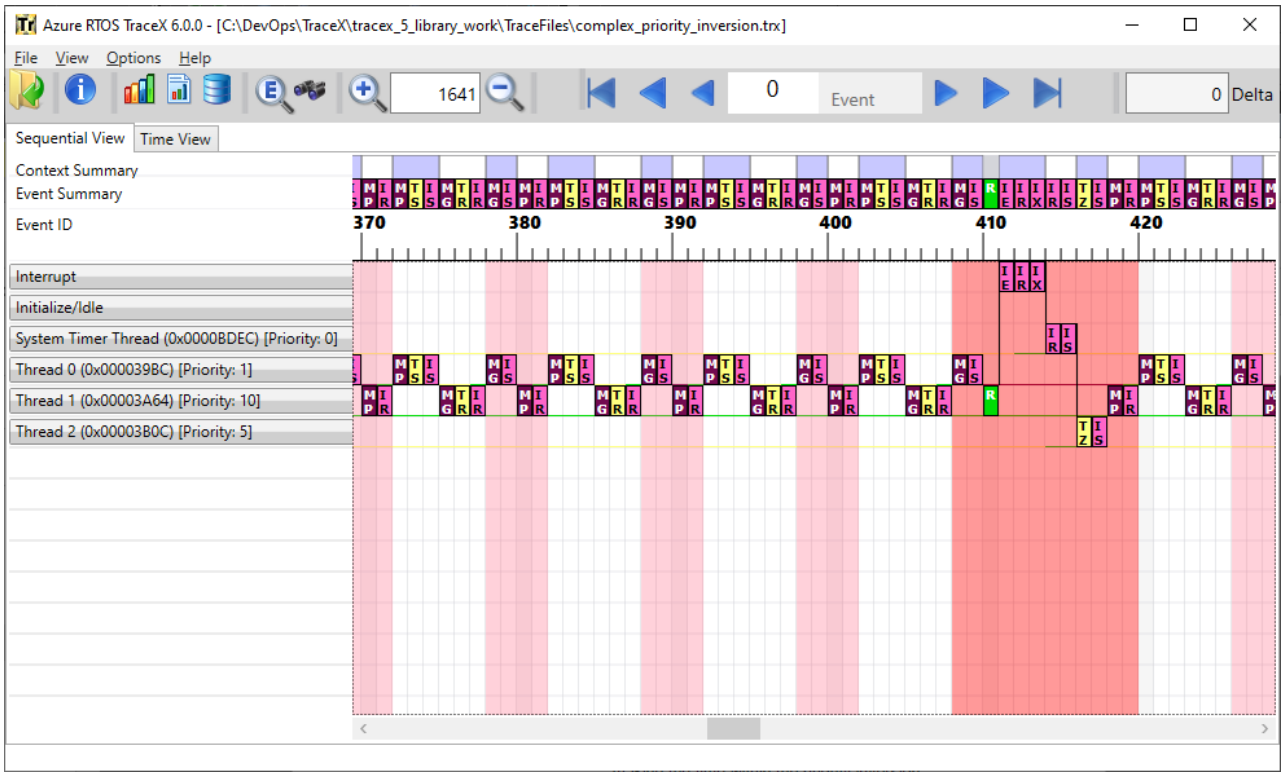
Delta Ticks

The delta ticks show that 5032 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.

Priority inversions

Azure RTOS 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,” since the system was setup to operate in this manner. In order to inform the user, Azure RTOS TraceX shows “deterministic” priority inversion ranges as a light salmon color.

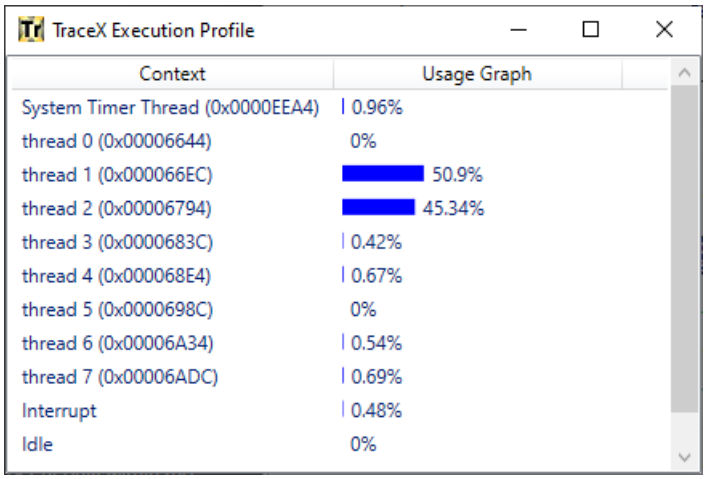
Azure RTOS TraceX also displays “un-deterministic” priority inversions. These priority inversions differ from the “deterministic” priority inversions in that another thread of a different priority level has executed in the middle of what was a “deterministic” priority inversion, thereby making the time within the priority inversion somewhat “un-deterministic.” This condition is often unknown to the user and can be very serious. In order to alert the user of this condition, Azure RTOS TraceX shows “un-deterministic” priority inversions as a brighter salmon color.



Deterministic and Non-Deterministic Priority Inversion

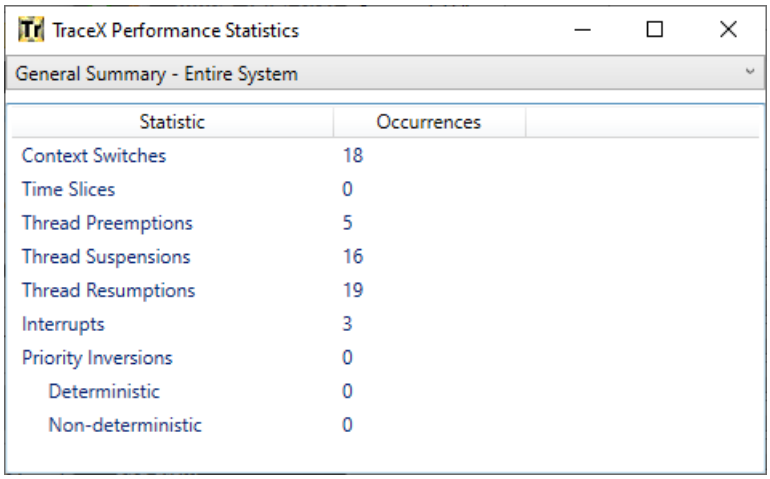
Execution profile

Azure RTOS TraceX provides a built-in execution profile report for all execution contexts within in the currently loaded trace file.



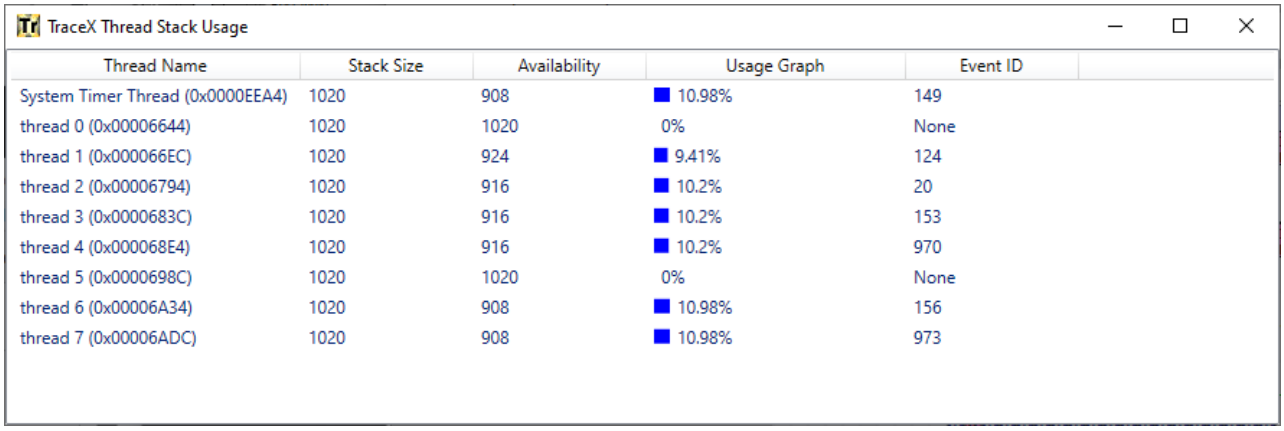
Performance statistics

Azure RTOS TraceX provides a built-in performance statistics report for the currently loaded trace file.

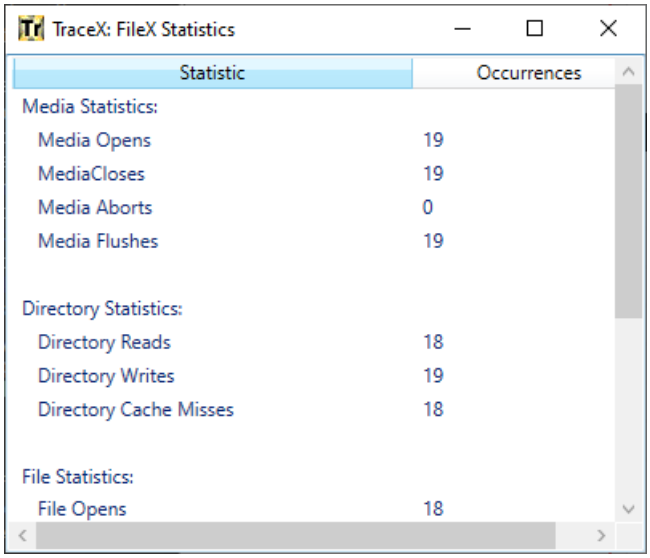


Thread stack usage

Azure RTOS TraceX provides a built-in stack usage report for all threads executing within in the currently loaded trace file.

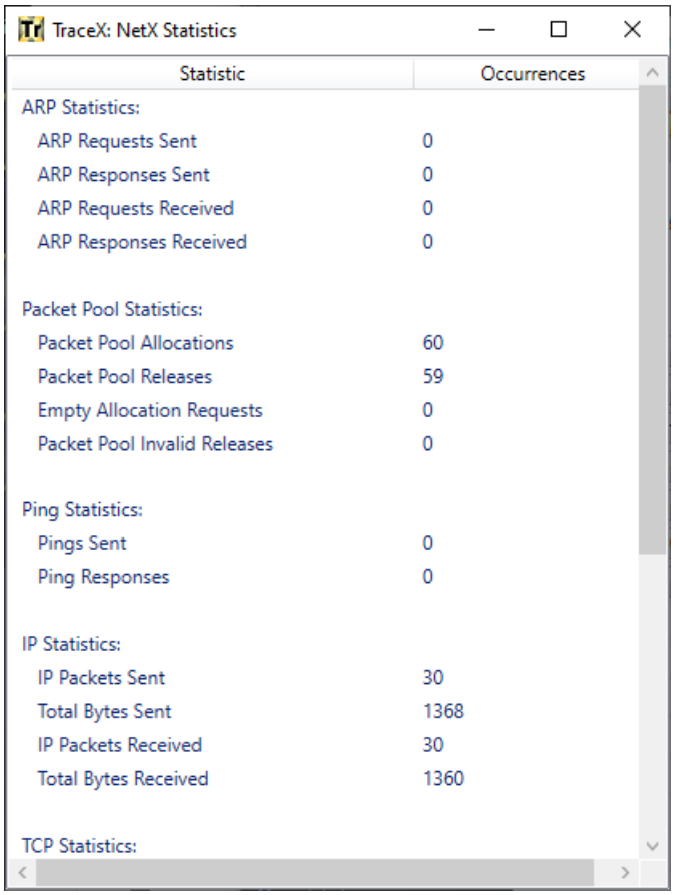


Azure RTOS TraceX presents the Azure RTOS FileX performance statistics of the currently loaded trace file. This information is displayed for the entire system–on all opened media objects.



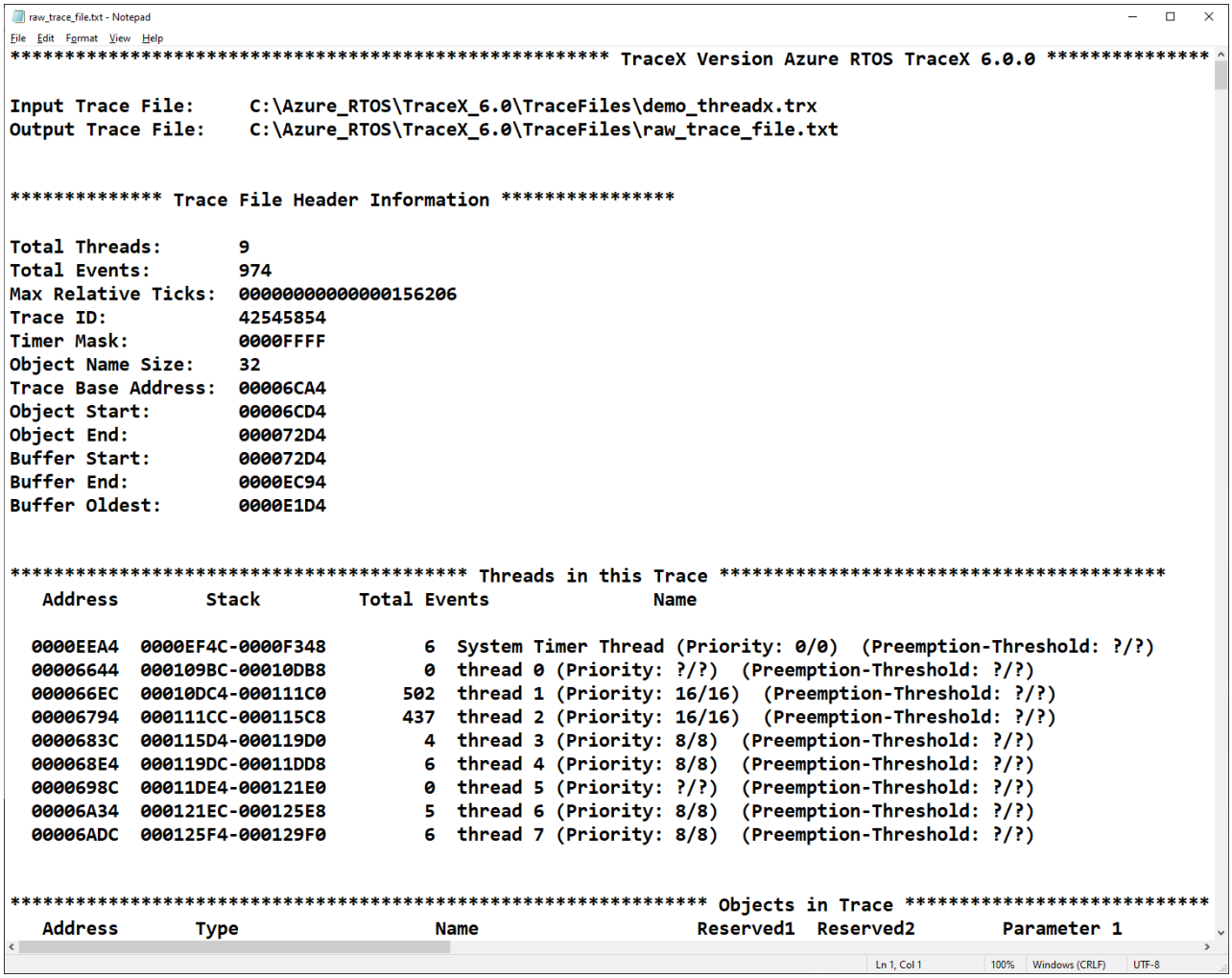
Azure RTOS NetX statistics

Azure RTOS TraceX also presents the NetX performance statistics of the currently loaded trace file. This information is displayed for the entire system.



Raw trace dump

Azure RTOS TraceX can build a raw trace file in text format and launch notepad to display it.



Please note that all timing and size figures listed are estimates and may be different on your development platform

About this guide

5/21/2020 • 2 minutes to read

This guide contains comprehensive information about Azure RTOS TraceX, the Microsoft Windows-based system analysis tool for Microsoft Azure RTOS.

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

Organization

- Chapter 1 - contains an basic overview of Azure RTOS TraceX and describes its relationship to real-time development.
- Chapter 2 - gives the basic steps to install and use Azure RTOS TraceX to analyze your application right out of the box.
- Chapter 3 - describes the main features of Azure RTOS TraceX.
- Chapter 4 - details performance analysis features of Azure RTOS TraceX.
- Chapter 5 - describes how to set up Azure RTOS ThreadX, Azure RTOS FileX, and Azure RTOS NetX in order to generate a trace buffer that is viewable by Azure RTOS TraceX.
- Chapter 6 - describes Azure RTOS TraceX events in detail.
- Chapter 7 - describes Azure RTOS FileX events in detail.
- Chapter 8 - describes Azure RTOS NetX events in detail.
- Chapter 9 - describes Azure RTOS USBX events in detail.
- Chapter 10 - describes creating custom user events in detail.
- Chapter 11 - describes the internal trace buffer in detail.
- Appendix A - Azure RTOS ThreadX port-specific file with its time-stamp source for gathering trace events.
- Appendix B - Azure RTOS 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 Azure RTOS TraceX binary files.
- Appendix D - Examples of dumping trace files from various development tools.

Guide Conventions

Italics - Typeface denotes book titles, emphasizes important words, and indicates variables.

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

> **NOTE**
> Indicates information of note.

Customer Support Center

Please submit a support ticket through the Azure Portal for questions or help using the steps here. Please supply us with the following information in an email message so we can more efficiently resolve your support request:

1. A detailed description of the problem, including frequency of occurrence and whether it can be reliably reproduced.
2. A detailed description of any changes to the application and/or Azure RTOS ThreadX that preceded the problem.
3. The contents of the *_tx_version_id* string found in the *tx_port.h* file of your distribution. This string will provide us valuable information regarding your run-time environment.
4. The contents in RAM of the *_tx_build_options* ULONG variable. This variable will give us information on how your Azure RTOS ThreadX library was built.

Chapter 1 - Introduction to Azure RTOS TraceX

5/21/2020 • 2 minutes to read

Azure RTOS 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.

TraceX Requirements

TraceX requires Windows XP (or above). The system should have a minimum of 192 MB 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 1 MB).
- 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.

Chapter 2 - Installation and use of Azure RTOS TraceX

7/20/2020 • 2 minutes to read

This chapter contains a description of various issues related to installation, setup, and usage of the Azure RTOS TraceX system analysis tool.

Product Distribution

The TraceX installer can be downloaded from https://aka.ms/azrtos-tracex-installer. Scroll to the bottom of the page, and select the "setup" program listed under **Assets**. The TraceX installer will install the TraceX application, the application dependencies, and several example trace data files which you can use to familiarize yourself with the TraceX application.

TraceX Installation Directory

TraceX, by default, is installed in the directory **c:/Azure_RTOS/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.

> **IMPORTANT**
>
> *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 2.5**. The installation dialogs are fairly straightforward, but it is worth noting that **Figure 2.3** shows the dialog for changing the default installation directory for TraceX.

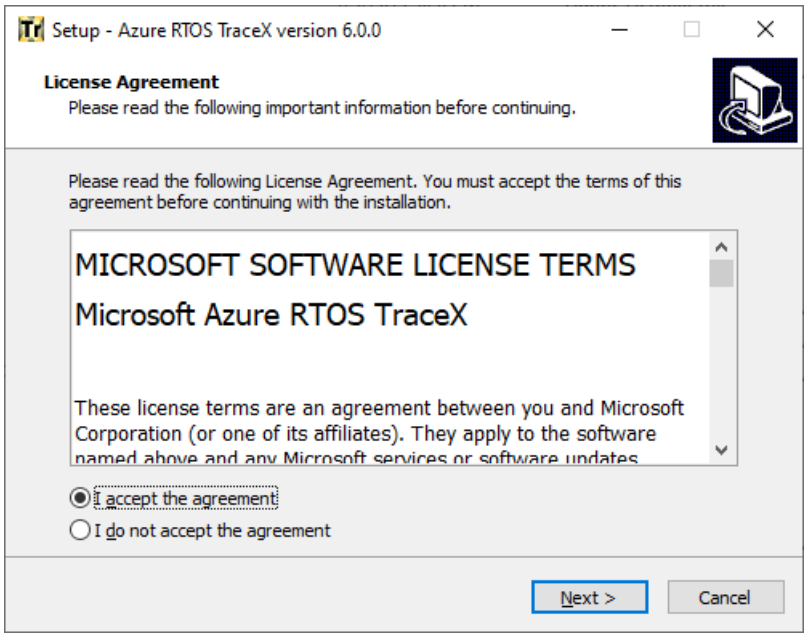


FIGURE 2.1

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 2.2**

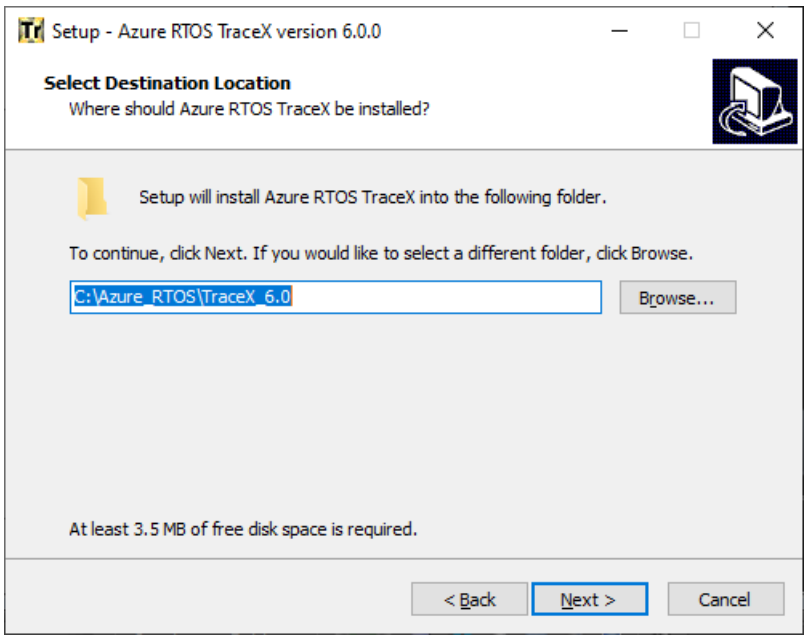


FIGURE 2.2

If the default installation path is okay, simply select the ***Next*** button to continue the installation, as shown in **Figure 2.3**

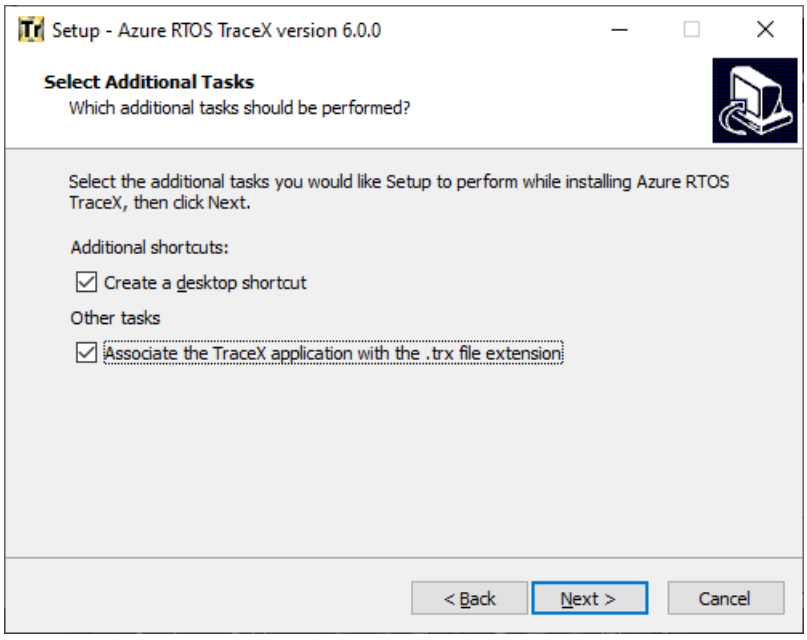


FIGURE 2.3

If everything is acceptable, select the ***Next*** button to continue the installation, as shown in **Figure 2.4**

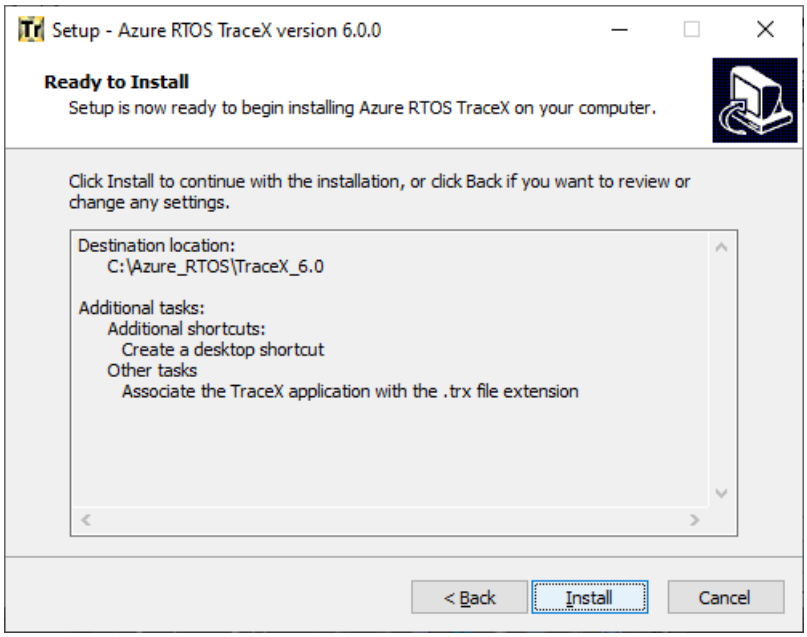


FIGURE 2.4

Select ***Install*** to install the application, and when the installation has finished selecting the ***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.

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.

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

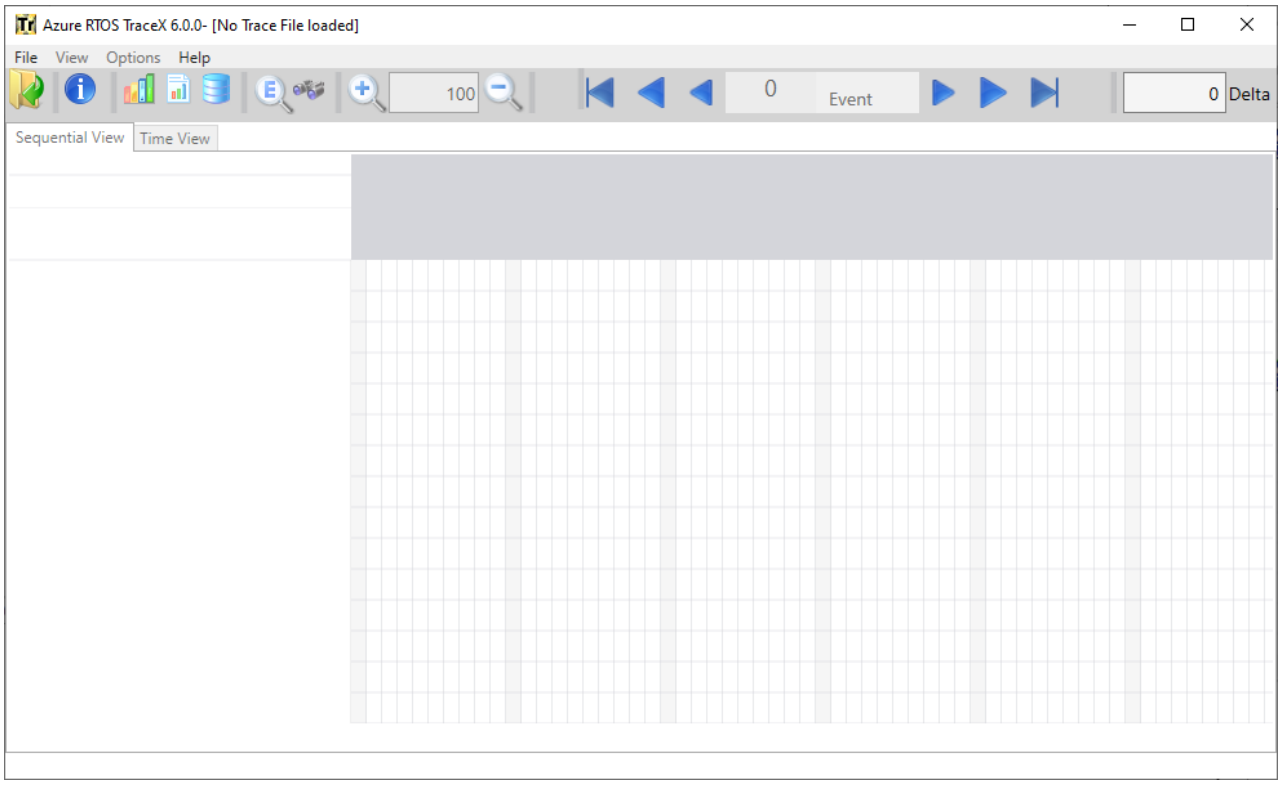


FIGURE 2.10

IMPORTANT

*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*.

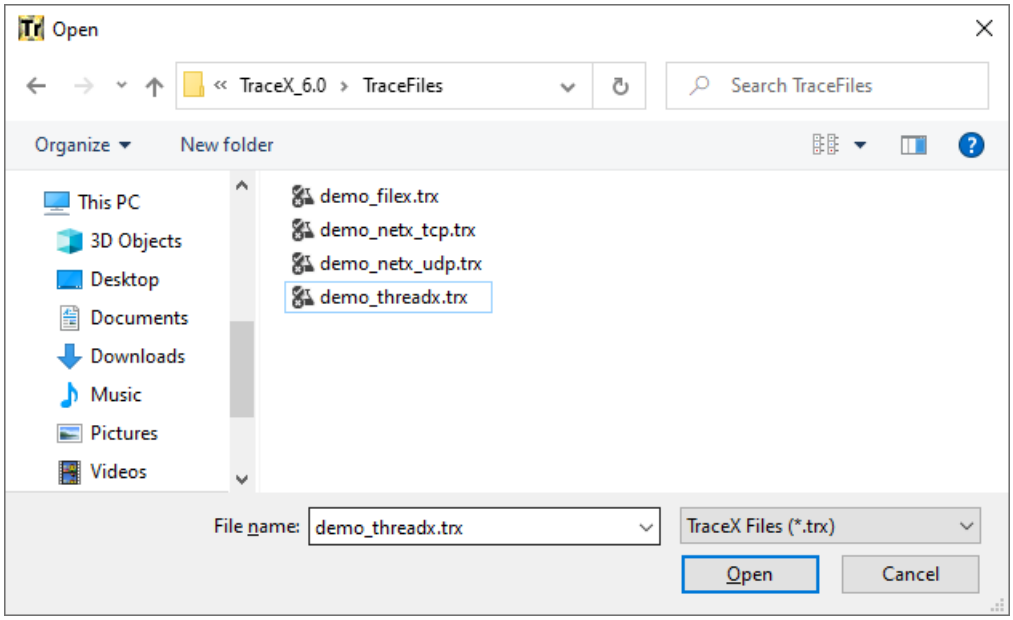


FIGURE 2.11

Chapter 3 - Description of Azure RTOS TraceX

7/20/2020 • 11 minutes to read

This chapter describes the overall functionality of the Azure RTOS TraceX system analysis tool, including the overall functionality of its GUI.

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***.

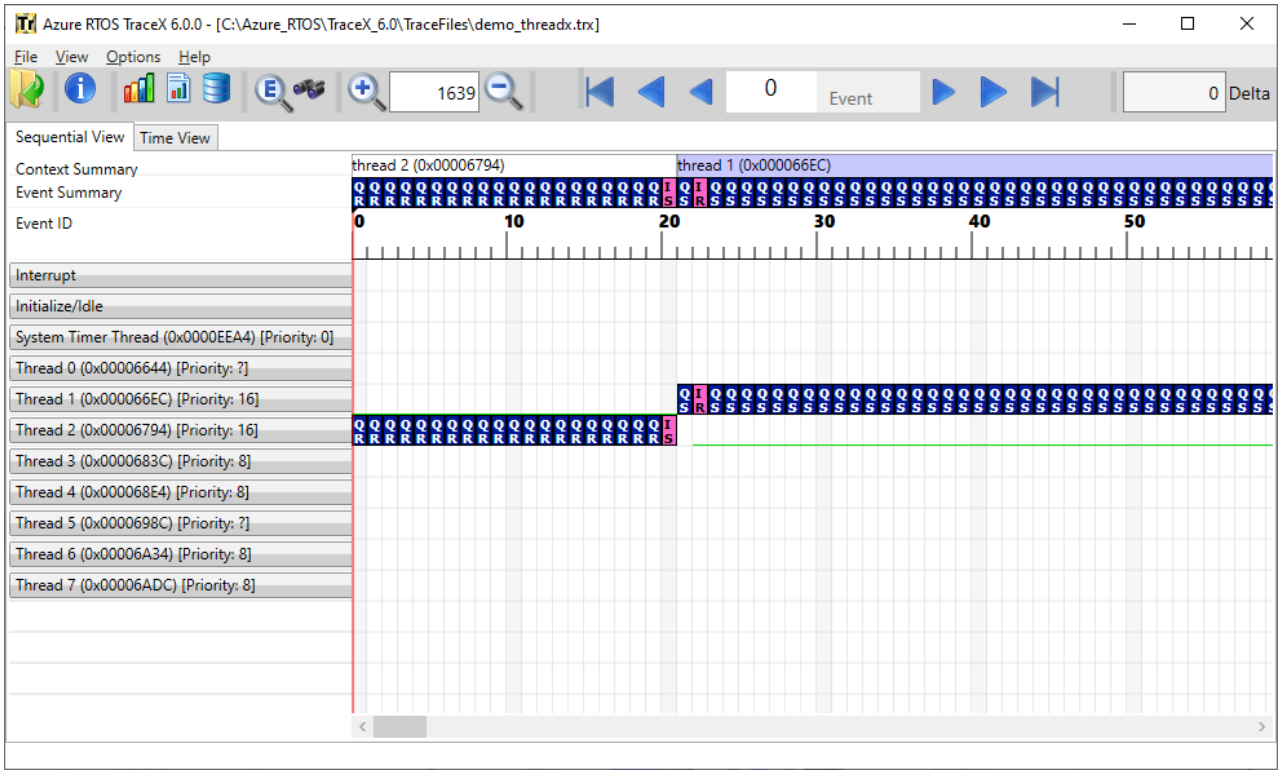


FIGURE 3.1

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 ***6.0.0*** is displaying the ***demo_threadx.trx*** trace file.

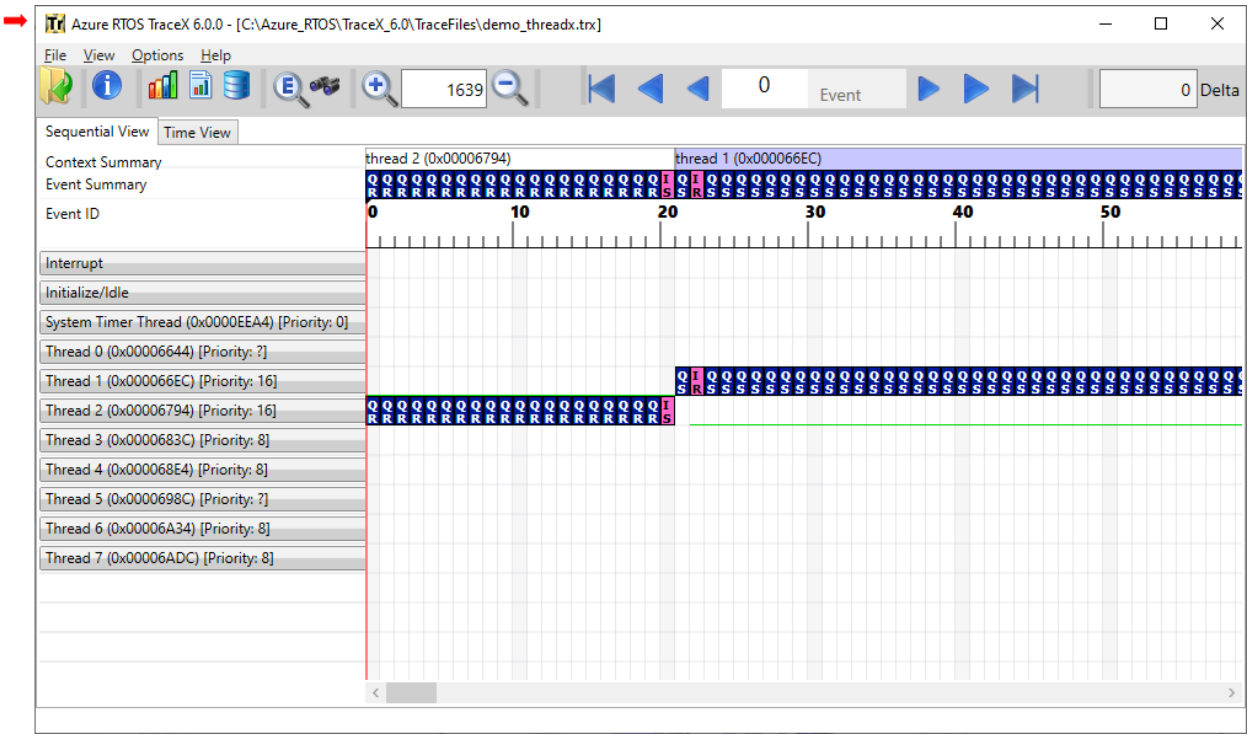


FIGURE 3.2

Tool Bar

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

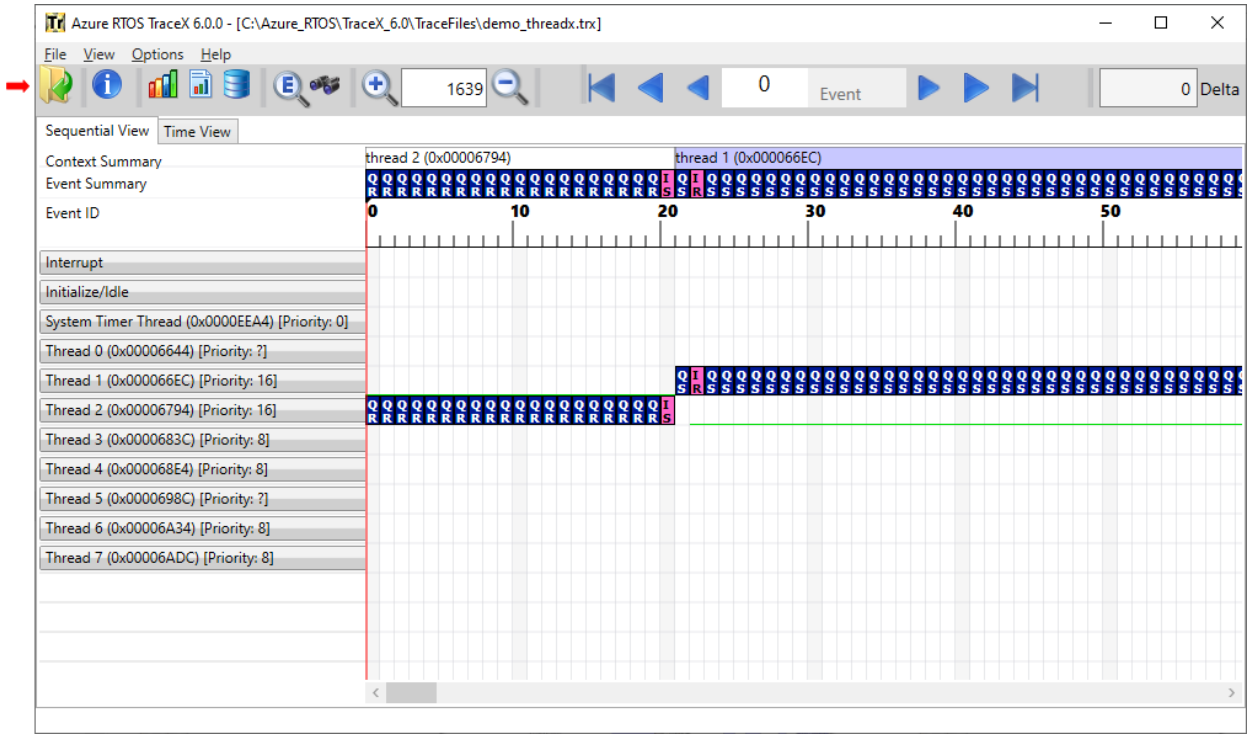
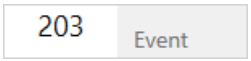


FIGURE 3.3

The TraceX tool bar buttons—from left to right—are defined as follows:

BUTTON	FUNCTION
	Open a trace file

BUTTON	FUNCTION
	Open this User Guide
	Generate execution profile
	Generate performance statistics
	Generate Thread Stack usage
	Display currently selected event
	Search for events
	Zoom in.
	Select percentage of display zoom, where 100% means the entire trace file is displayed within the current view.
	Zoom out.
	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.

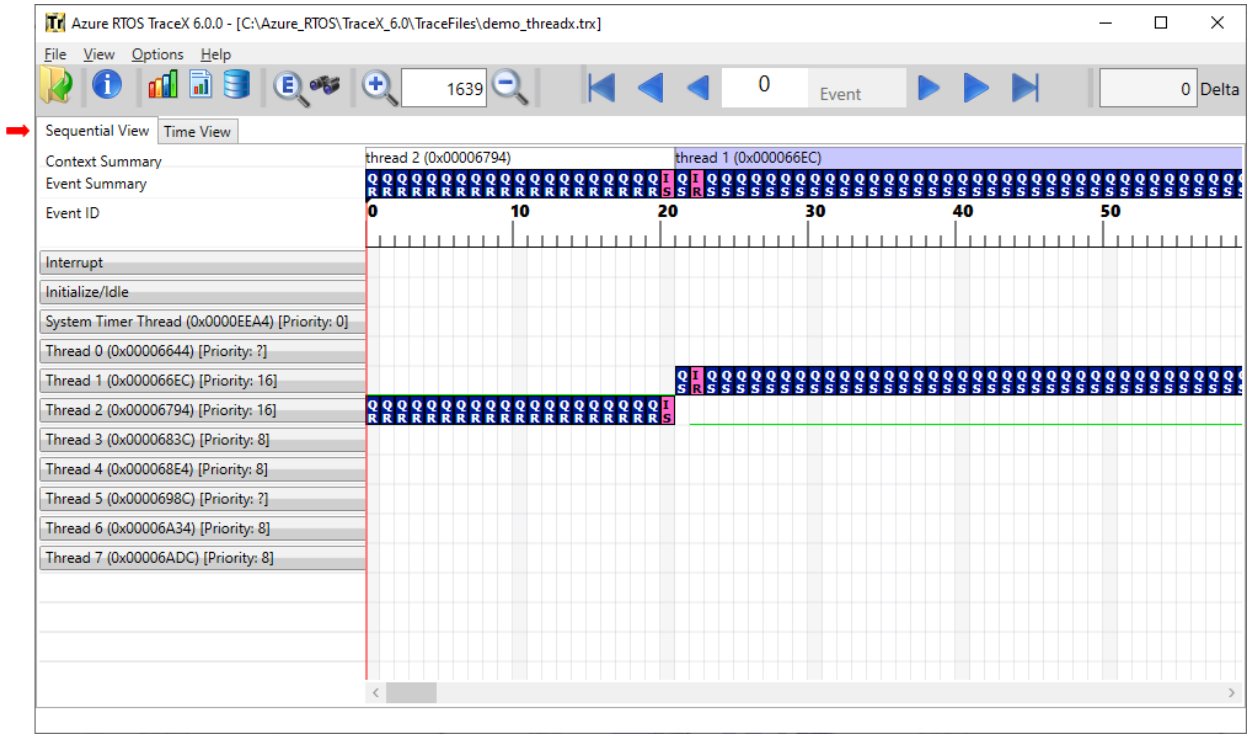


FIGURE 3.4

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 im../mediately 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.

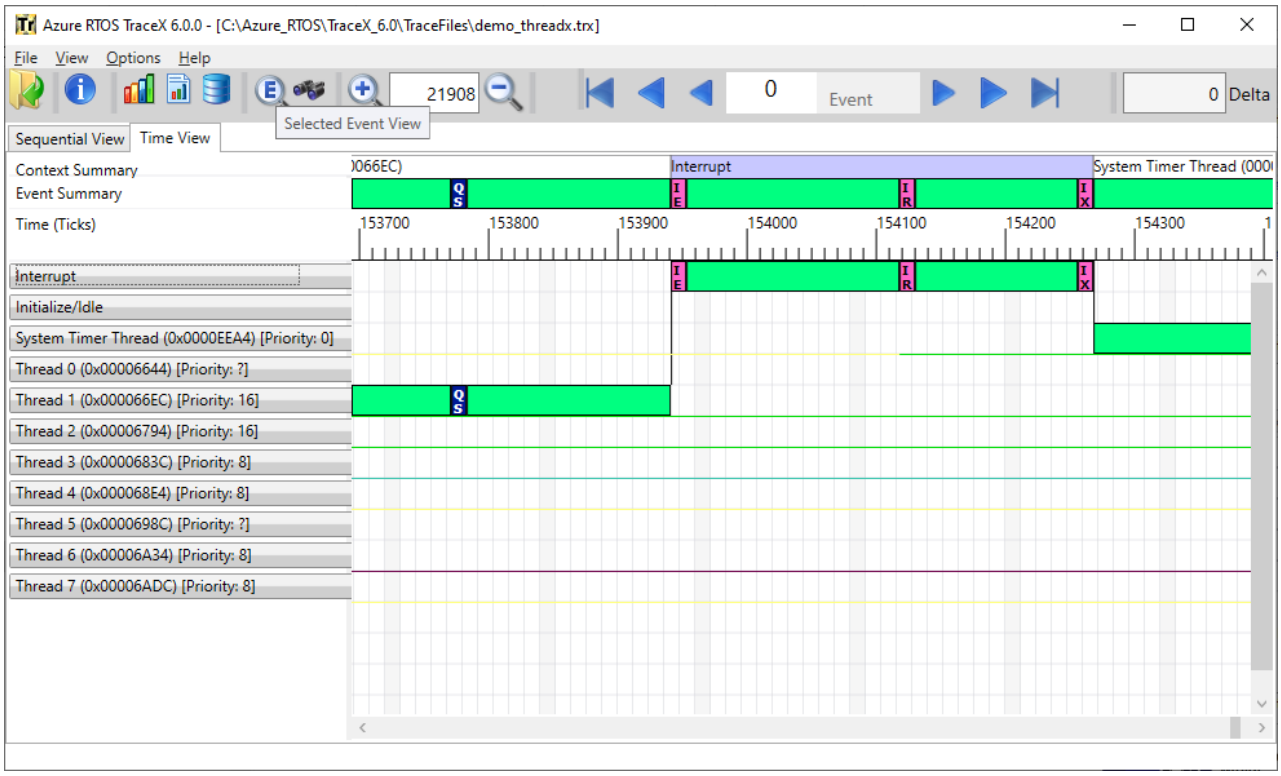


FIGURE 3.5

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.

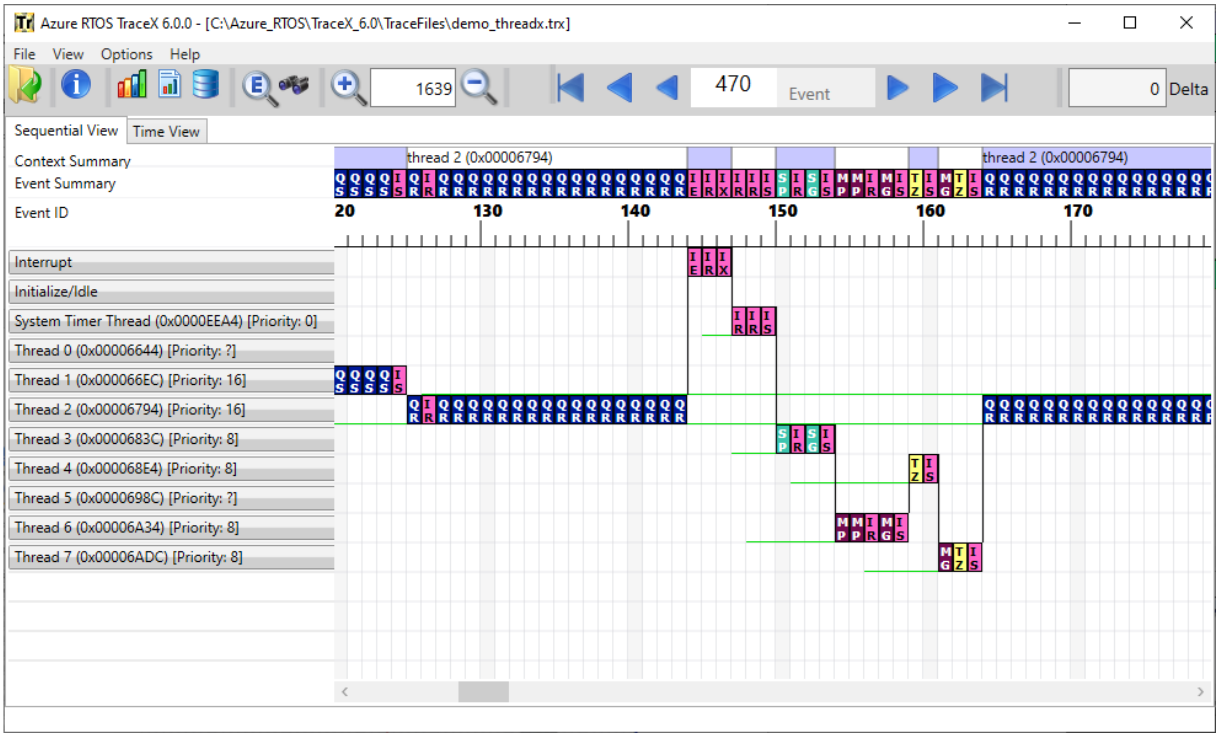


FIGURE 3.6

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 tow context entries are always the ***Interrupt*** and ***Initialize/Idle*** contexts. ***Interrupt*** context represents all system events made from Interrupt Service Routines (IRS). ***Initialize/Idle*** context represents two contexts in ThreadX. Events that occur during ***tx_application_define***, are ***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.

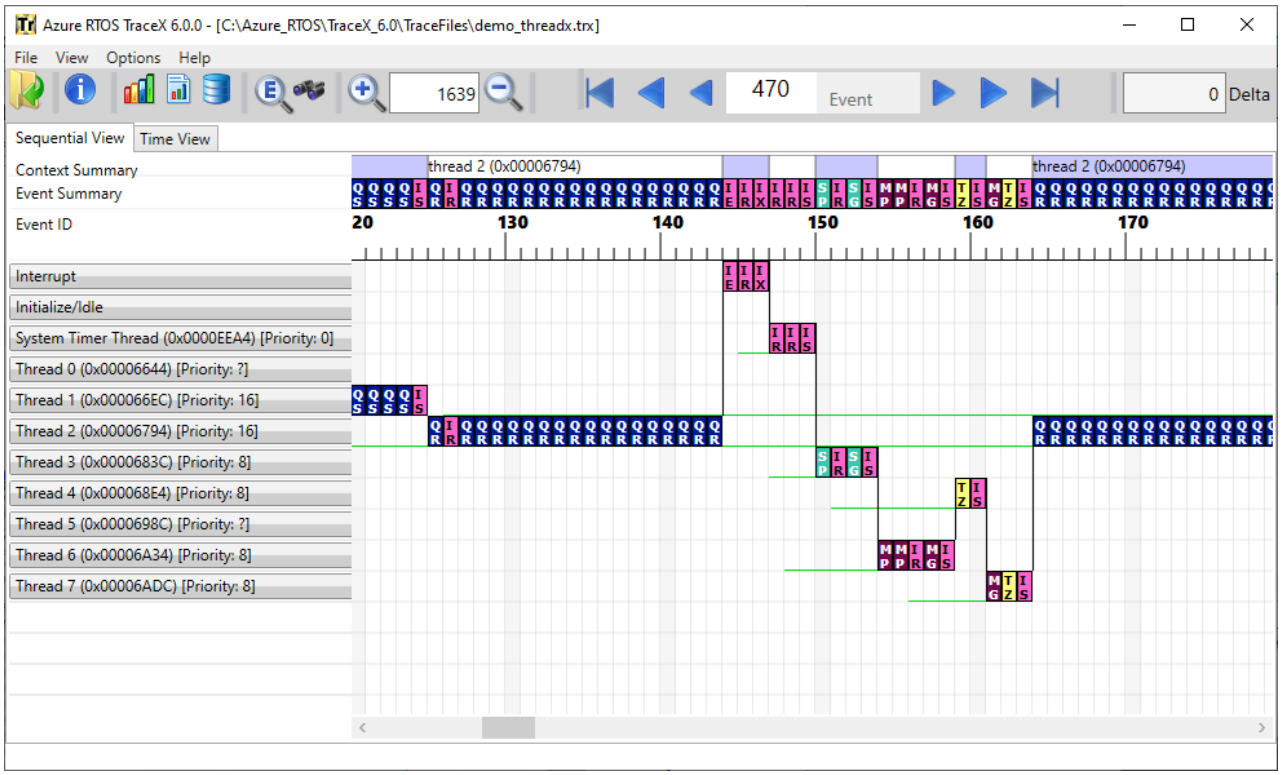


FIGURE 3.7

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***.

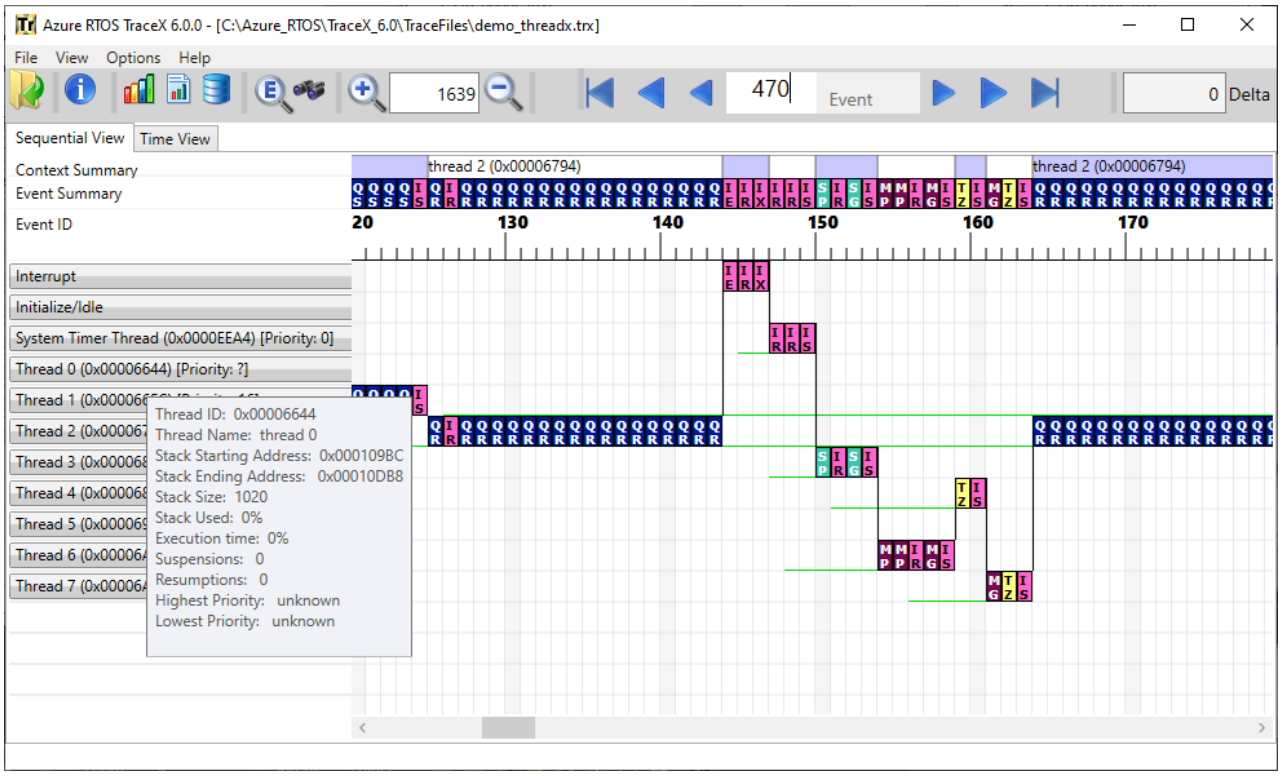


FIGURE 3.8

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.

Selecting ***Move to top*** results in the ***thread 3*** context being moved to the top of the context list, as shown in Figure 3.9.

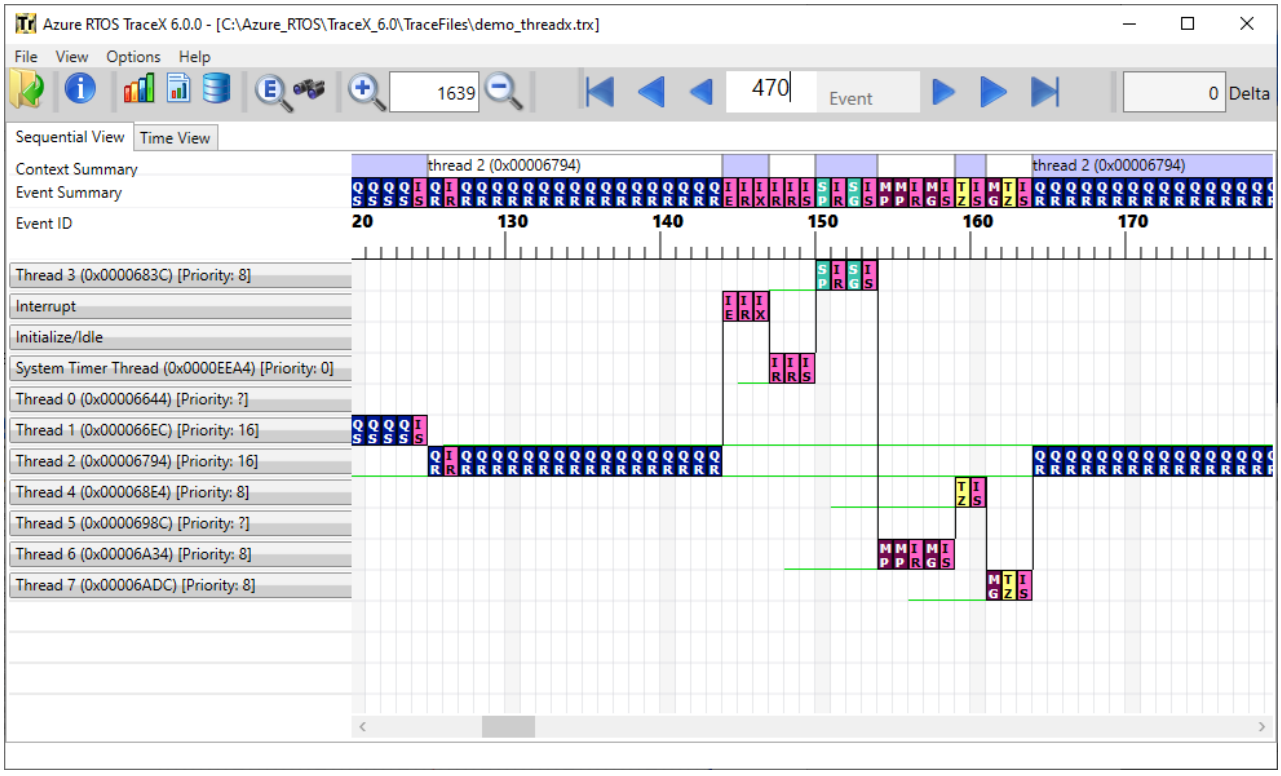


FIGURE 3.9

Thread Status Information

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 3.10** the green line on the ***System Timer Thread's*** context starting at event 147 shows that the ***System Timer Thread*** is ready. Prior to event 147 and after event 154, the absence of the green line indicates that the ***System Timer Thread*** is ready. Prior to event 147 and after event 154, the absence of the green line indicates that the ***System Timer Thread*** is suspended.

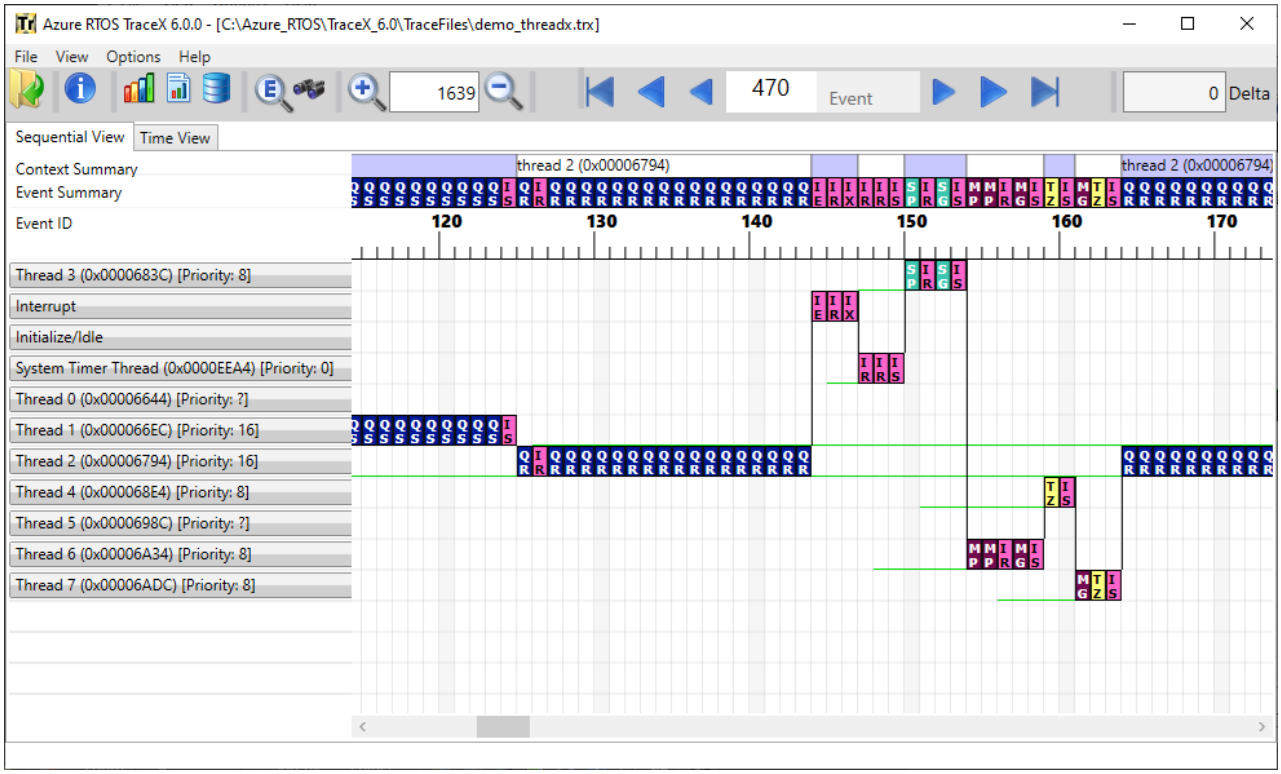


FIGURE 3.10

There are three modes of thread status display, available via the ***Options -> Status Lines*** menu. 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

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 143 in the demonstration ***demo_threadx.trx*** trace file is shown in **Figure 3.11**:

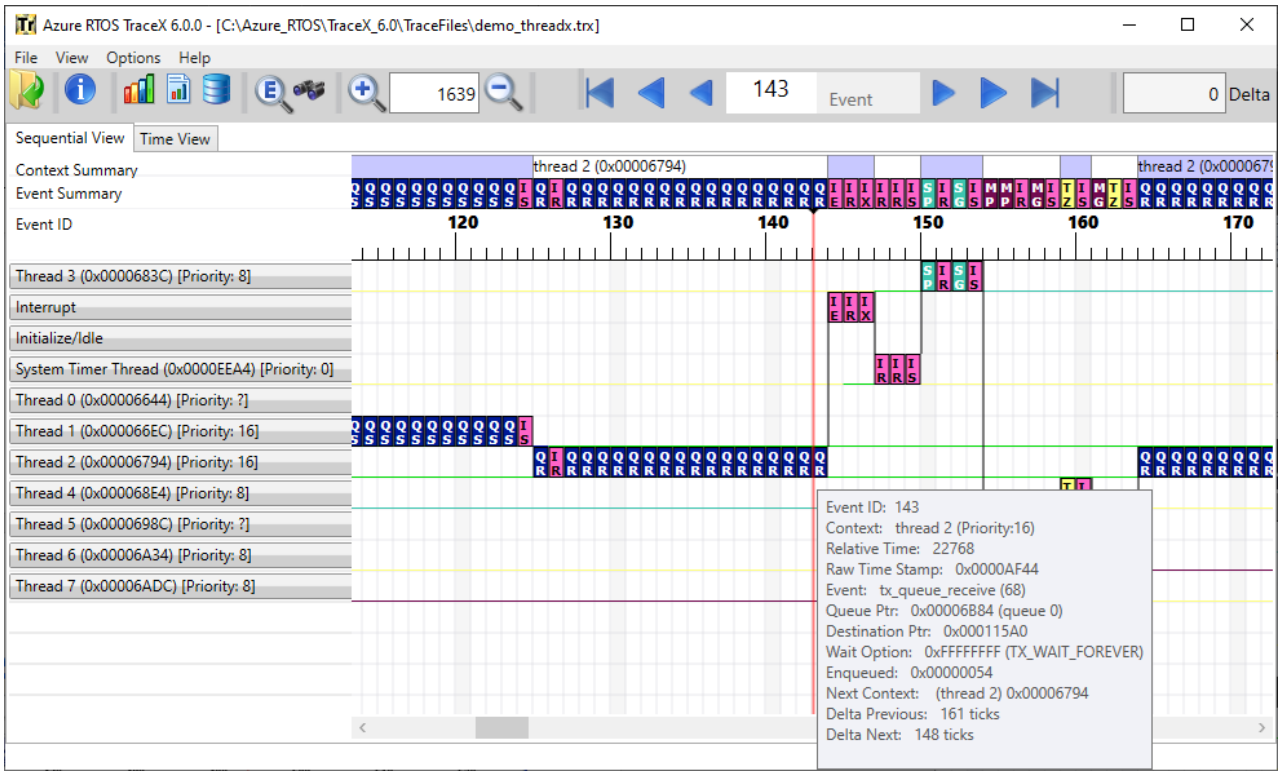


FIGURE 3.11

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 143 is shown in **Figure 3.12**:

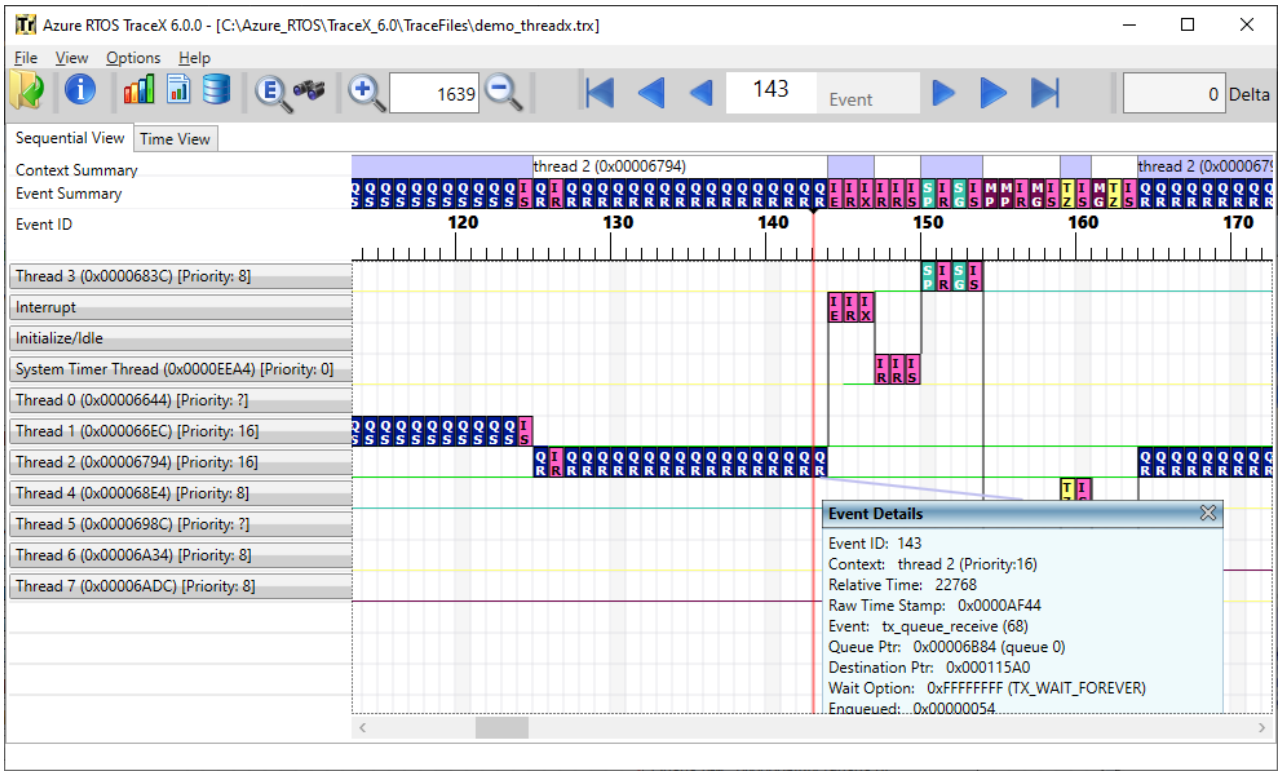


FIGURE 3.12

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. After selected, TraceX displays the currently selected event in a stand-alone window and refreshes this window whenever another event is selected.

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 of 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 3.13**. 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 3.14** shows the search dialog.

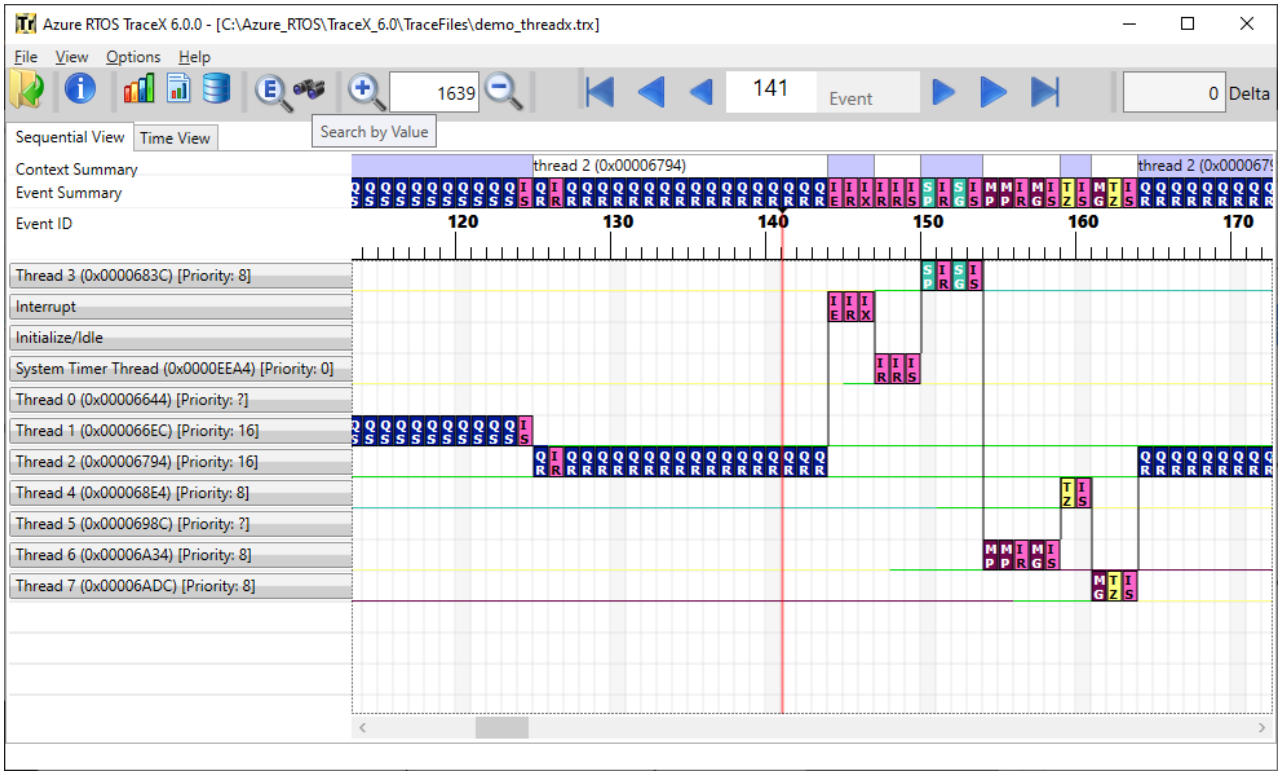


FIGURE 3.13

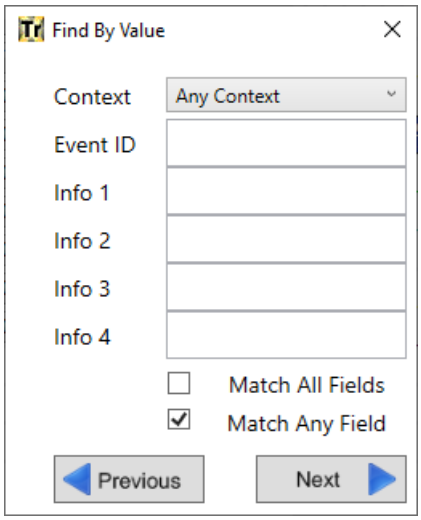


FIGURE 3.14

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 3.15** shows the ***demo_threadx.trx*** file zoomed out so that 100% of the trace file is shown.

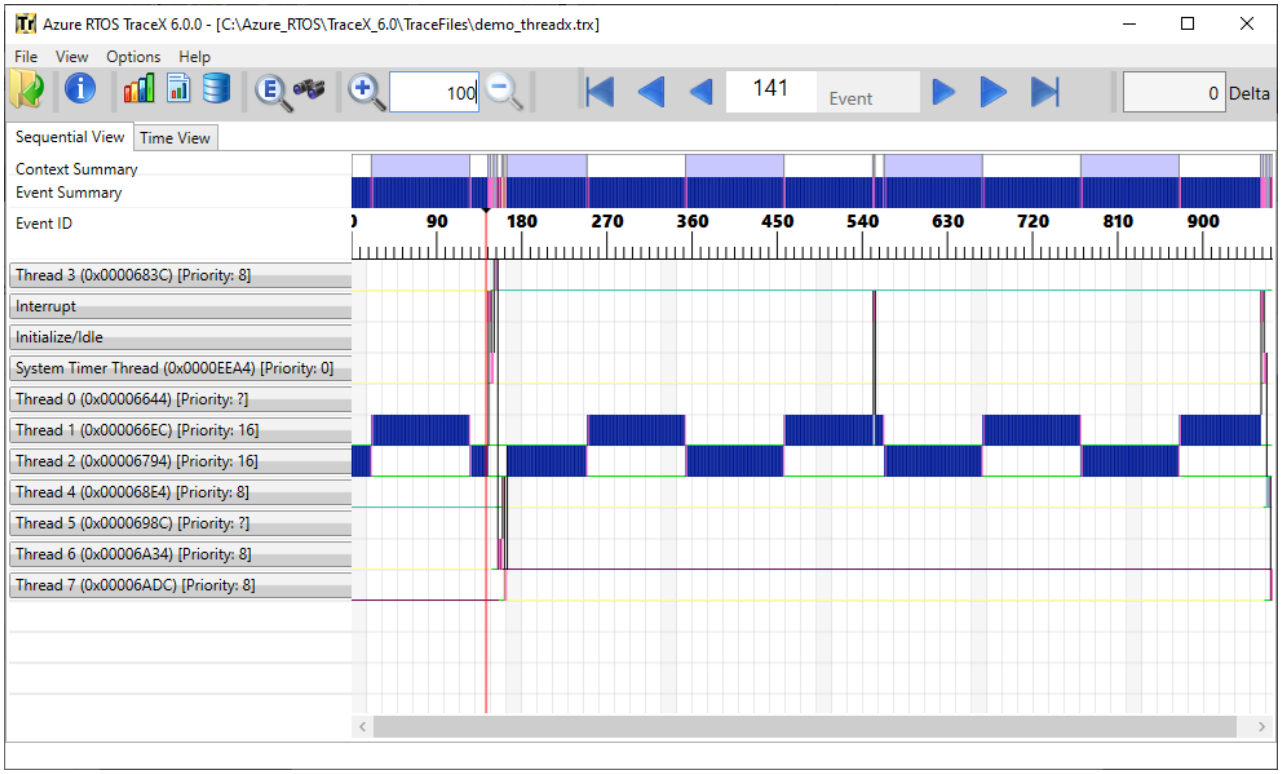


FIGURE 3.15

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 3.16**.

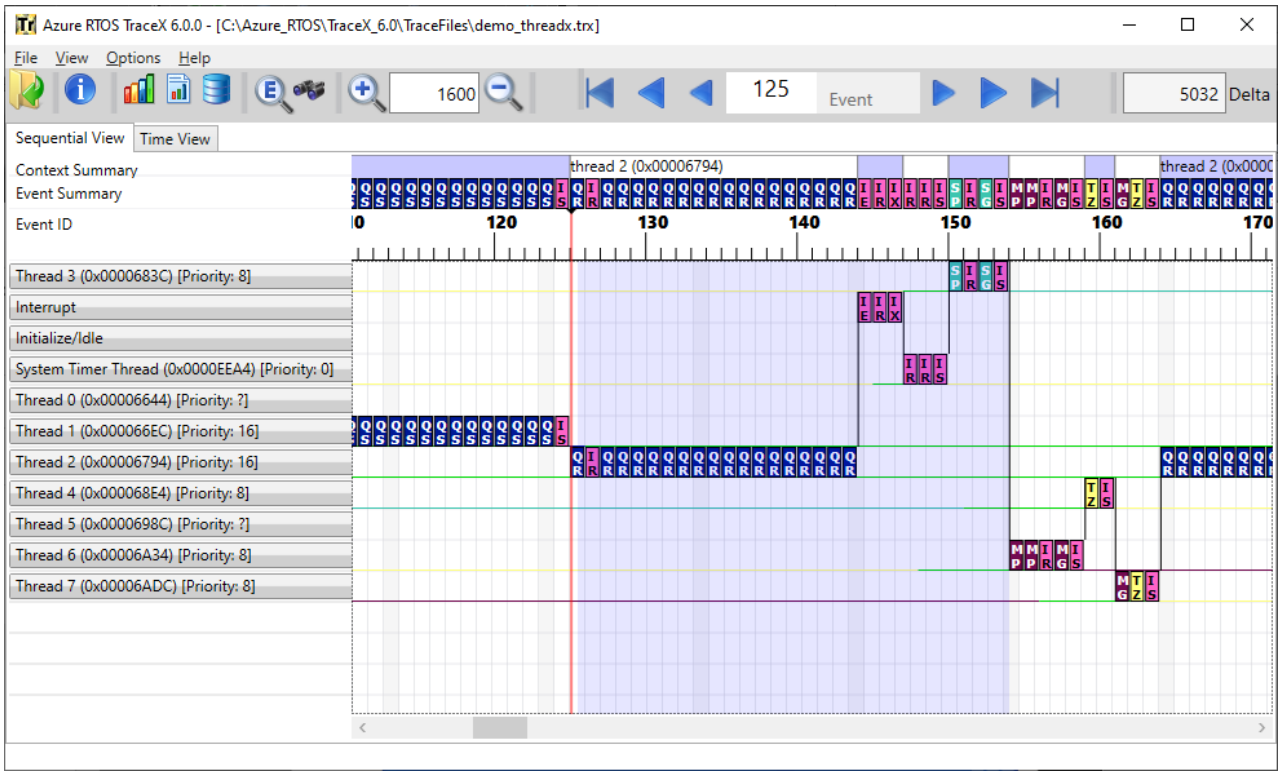


FIGURE 3.16

The delta ticks shown in **Figure 3.16** show that 5032 ticks have elapsed between event 125 and event 154. 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).

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 *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 *non-deterministic* priority inversions as a brighter salmon color. **Figure 3.17** shows both *deterministic* and *non-deterministic* priority inversions.

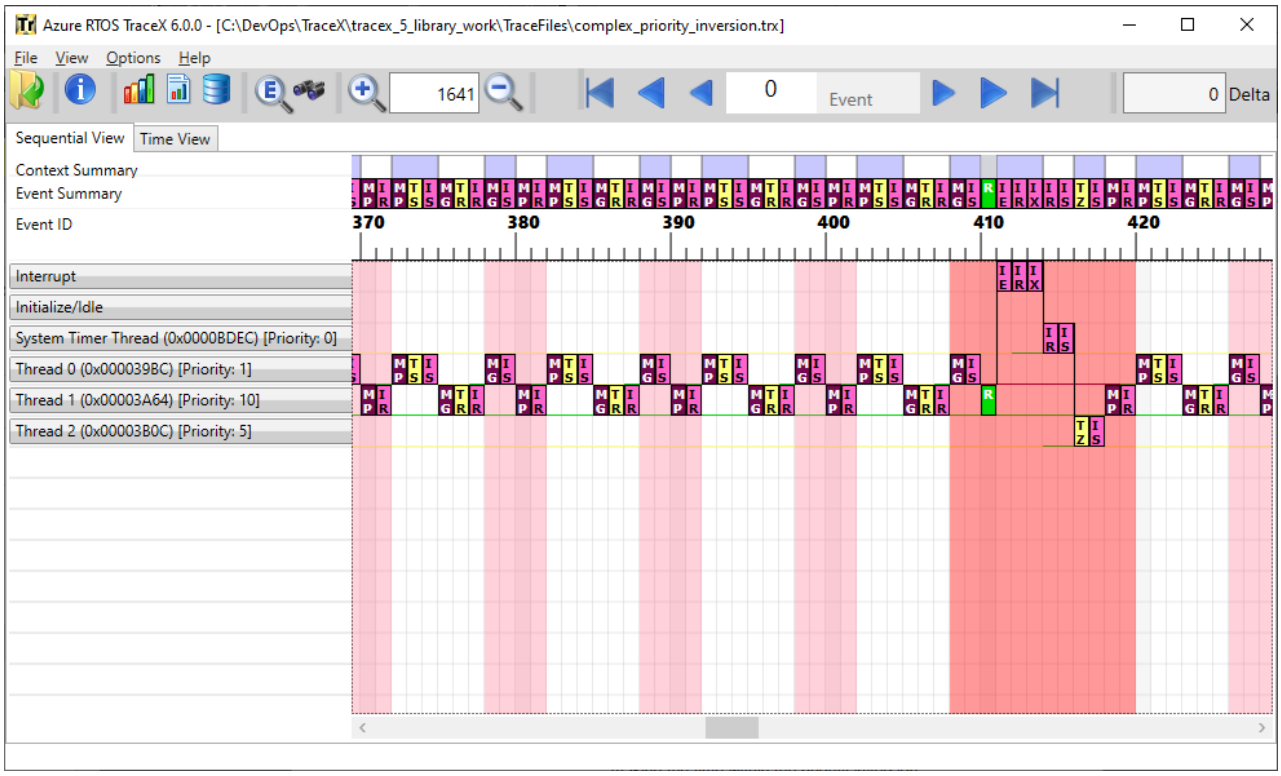


FIGURE 3.17

Figure 3.17 shows a *deterministic* priority inversion from event 398 through event 402. In this range, the higher-priority ***thread 0*** blocks on a mutex owned by a lower-priority ***thread 1***. At event 402, ***thread 1*** releases the mutex and thus ends the priority inversion.

The brighter shaded area shows a *non-deterministic* priority inversion between event 408 through event 420. What makes this *non-deterministic* is that while ***thread 1*** holds the mutex that higher-priority ***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.

Chapter 4 - Azure RTOS TraceX performance analysis

7/20/2020 • 4 minutes to read

This chapter describes the Azure RTOS TraceX performance analysis tool:

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.

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 4.1**.

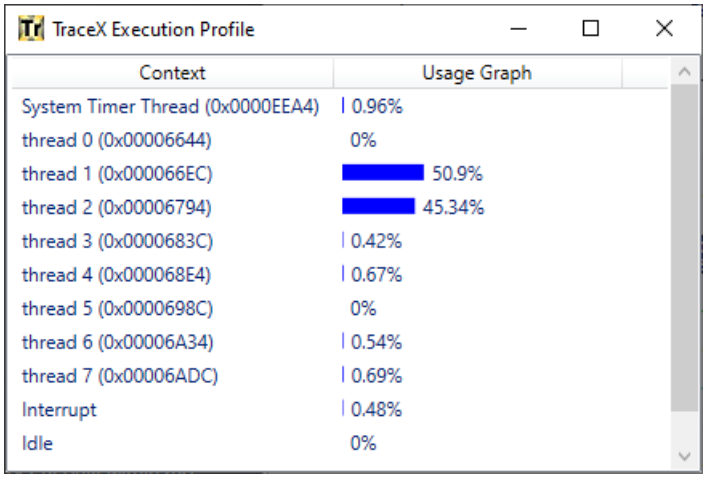


FIGURE 4.1

The example shown in **Figure 4.1** indicates that nearly 45% of the processing time is inside of ***thread 2*** and nearly 51% 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 4.2**.

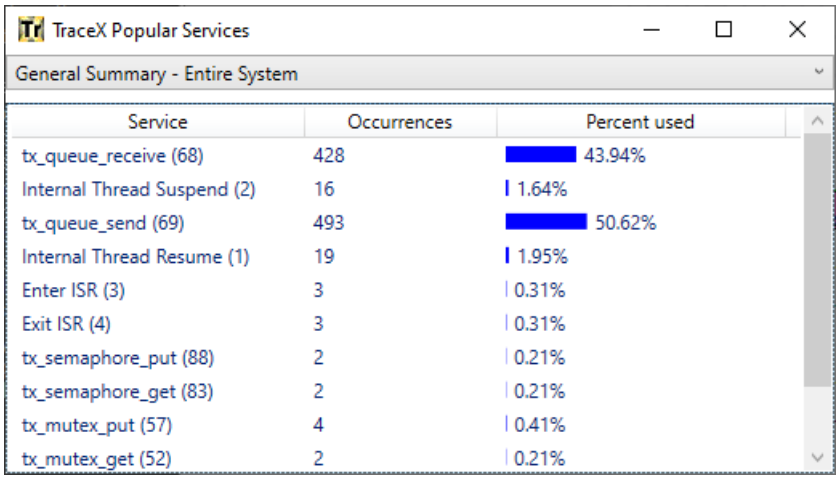


FIGURE 4.2

The example shown in **Figure 4.2** indicates that ***tx_queue_send*** and ***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.

Specific threads can be selected for this analysis by using the drop down selection list at the top of this window. **Figure 4.3** shows this analysis for ***thread 3***.

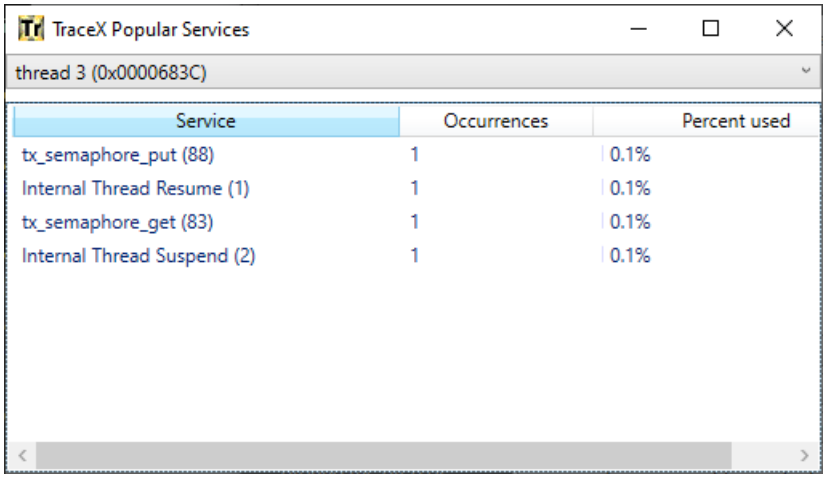


FIGURE 4.3

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 4.4**.

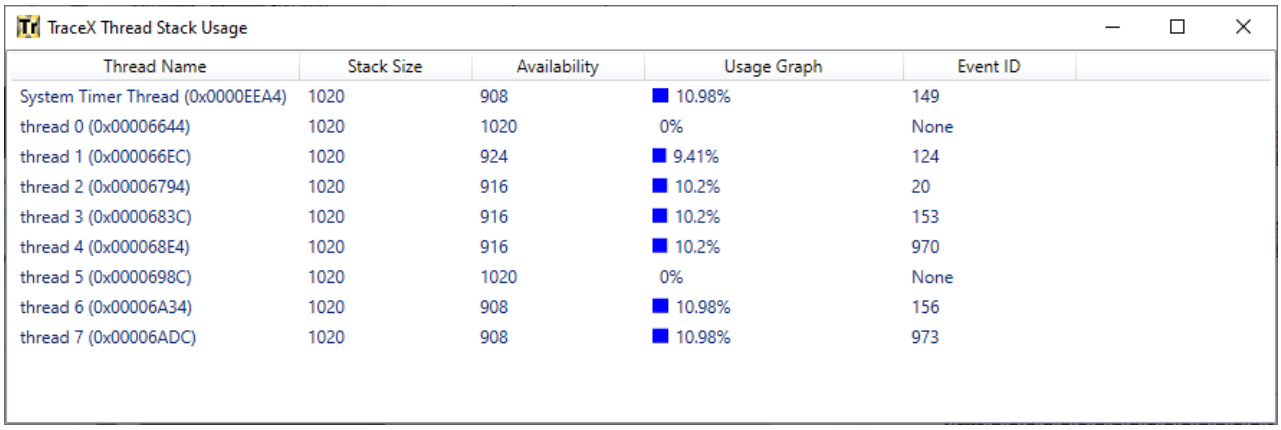


FIGURE 4.4

The example shown in **Figure 4.4** 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 4.5**.

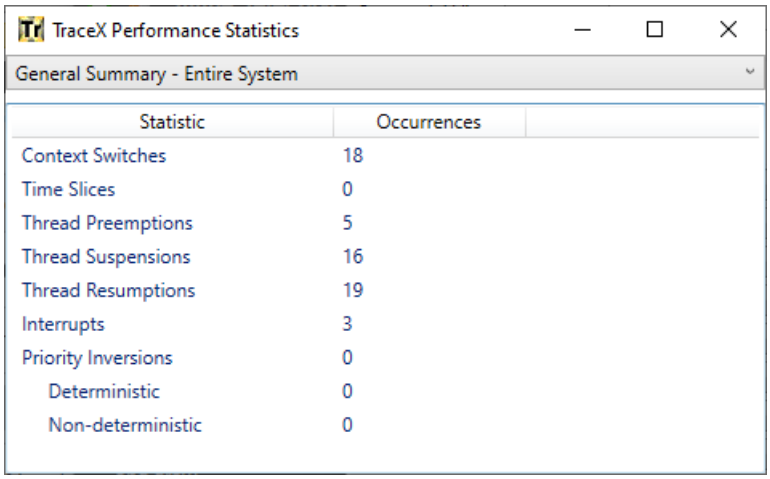


FIGURE 4.5

The example shown in **Figure 4.5** indicates that there were 18 context switches in this trace file, as well as five thread preemptions, 16 thread suspensions, 19 thread resumptions, and three interrupts. There were no priority inversions found in this trace file. Notice there are two categories of priority inversions, namely, *deterministic* and *nondeterministic*. Deterministic priority inversions are priority inversion in which a thread is blocked on a mutex owned by a lower priority thread. An nondeterministic 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.

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 performance statistics of the sample FileX demonstration trace are shown in **Figure 4.6**.

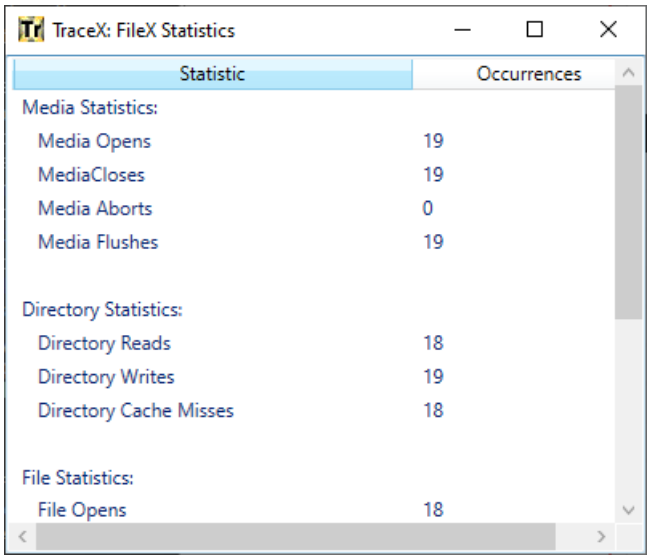


FIGURE 4.6

The example shown in **Figure 4.9** indicates there were 19 ../media opens, 19 ../media closes, 19 ../media flushes, 18 directory reads, 19 directory writes, and 18 directory cache misses. Additonal information can be viewed by scrolling down in the statistics window.

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 4.7**.

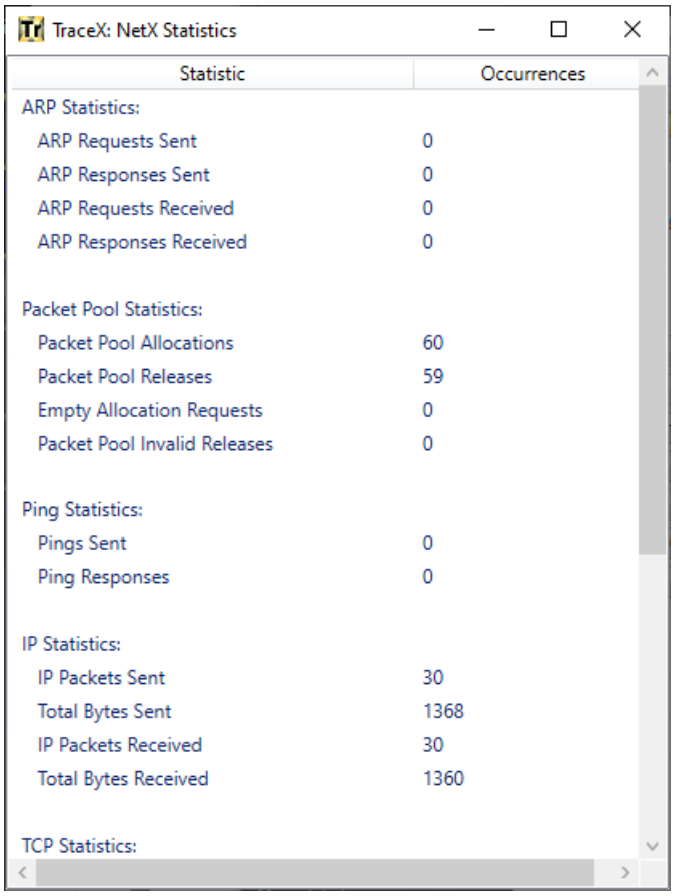


FIGURE 4.7

The example shown in **Figure 4.7** indicates there were no ARP, Ping, or UDP events, but there were 30 IP packets sent, 1,368 IP bytes sent, 30 IP packets received, and 1,360 IP 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 4.8** shows the trace file information for the standard *demo_threadx.trx* trace file.

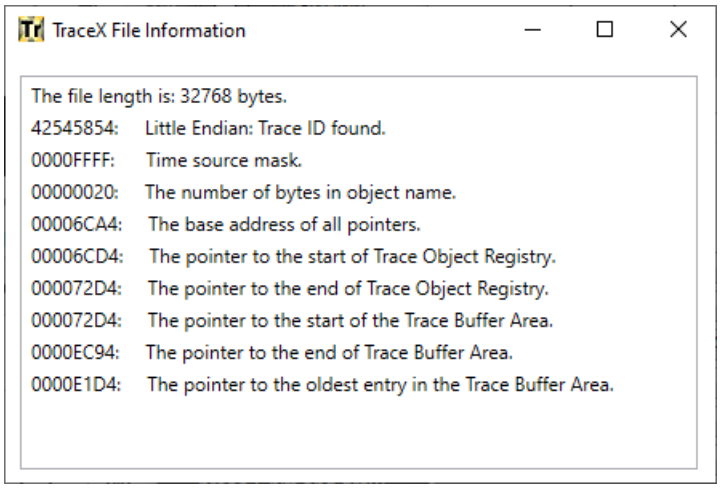


FIGURE 4.8

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 4.9** shows the raw trace file dump for the standard *demo_threadx.trx* trace file.

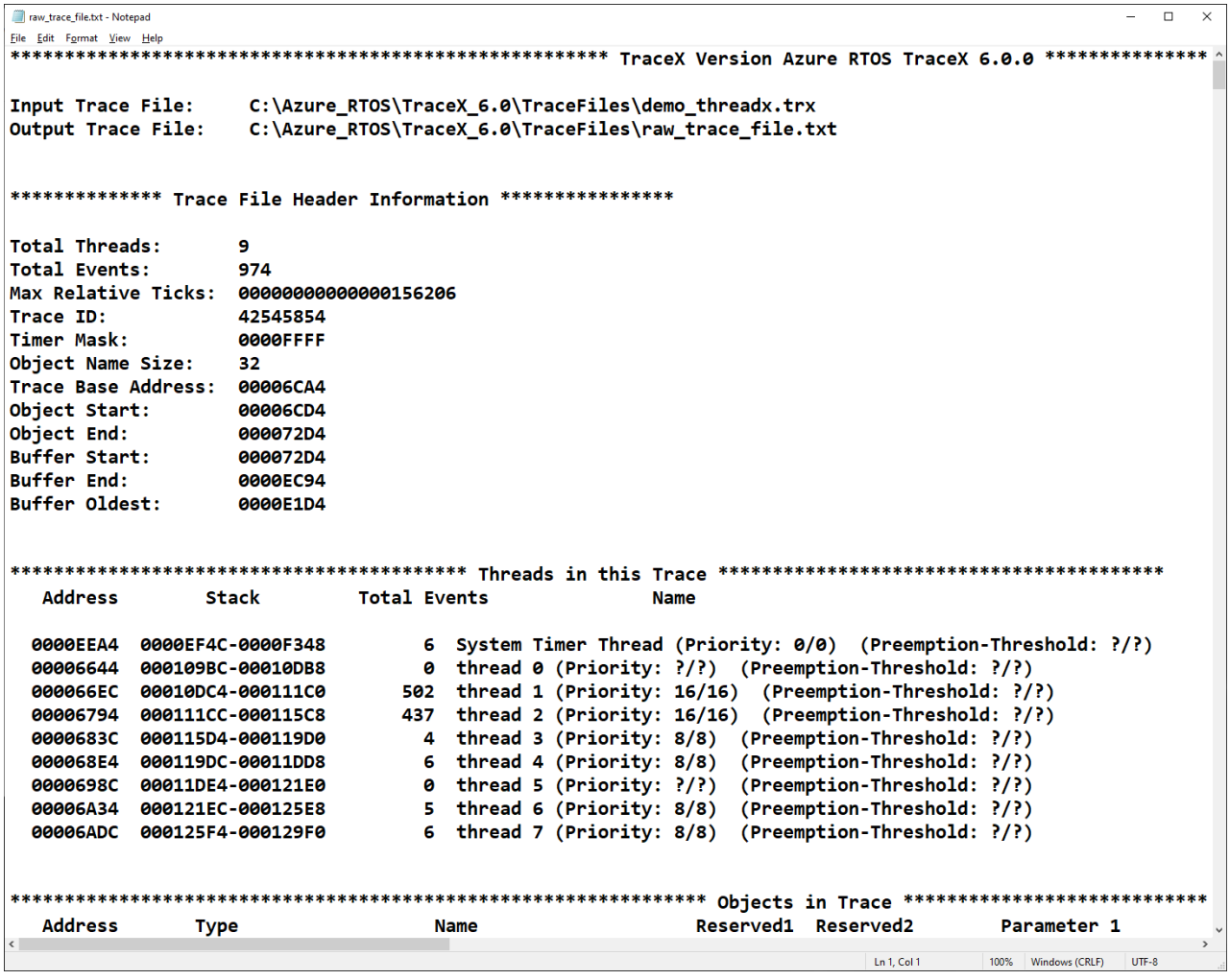


FIGURE 4.9

Chapter 5 - Generating trace buffers

5/21/2020 • 8 minutes to read

This chapter contains a description about how to build a Azure RTOS TraceX event buffer and also describes the underlying format of the buffer.

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:

```
#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 timestamp definition:

```
#ifndef TX_TRACE_TIME_SOURCE
#define TX_TRACE_TIME_SOURCE ((ULONG) 0x0x13000004)
#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.*

> **IMPORTANT**
>
> *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

```
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.

> **IMPORTANT**
>
> 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

```
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  vent_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.

> **IMPORTANT**
>
> 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 oring together the appropriate constants. Valid constants for this variable are defined as follows:

```
TX_TRACE_ALL_EVENTS                  0x000007FF
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                  0x00007800
FX_TRACE_INTERNAL_EVENTS             0x00000800
FX_TRACE_MEDIA_EVENTS                0x00001000
FX_TRACE_DIRECTORY_EVENTS            0x00002000
FX_TRACE_FILE_EVENTS                 0x00004000
NX_TRACE_ALL_EVENTS                  0x00FF8000
NX_TRACE_INTERNAL_EVENTS             0x00008000
NX_TRACE_ARP_EVENTS                  0x00010000
NX_TRACE_ICMP_EVENTS                 0x00020000
NX_TRACE_IGMP_EVENTS                 0x00040000
NX_TRACE_IP_EVENTS                   0x00080000
NX_TRACE_PACKET_EVENTS               0x00100000
NX_TRACE_RARP_EVENTS                 0x00200000
NX_TRACE_TCP_EVENTS                  0x00400000
NX_TRACE_UDP_EVENTS                  0x00800000
UX_TRACE_ALL_EVENTS                  0x7F000000
UX_TRACE_ERRORS                      0x01000000
UX_TRACE_HOST_STACK_EVENTS           0x02000000
UX_TRACE_DEVICE_STACK_EVENTS         0x04000000
UX_TRACE_HOST_CONTROLLER_EVENTS      0x08000000
UX_TRACE_DEVICE_CONTROLLER_EVENTS    0x10000000
UX_TRACE_HOST_CLASS_EVENTS           0x20000000
UX_TRACE_DEVICE_CLASS_EVENTS         0x40000000
```

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.

> **IMPORTANT**
>
> 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:

```
TX_TRACE_ALL_EVENTS                 0x000007FF
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                 0x00007800
FX_TRACE_INTERNAL_EVENTS            0x00000800
FX_TRACE_MEDIA_EVENTS               0x00001000
FX_TRACE_DIRECTORY_EVENTS           0x00002000
FX_TRACE_FILE_EVENTS                0x00004000
NX_TRACE_ALL_EVENTS                 0x00FF8000
NX_TRACE_INTERNAL_EVENTS            0x00008000
NX_TRACE_ARP_EVENTS                 0x00010000
NX_TRACE_ICMP_EVENTS                0x00020000
NX_TRACE_IGMP_EVENTS                0x00040000
NX_TRACE_IP_EVENTS                  0x00080000
NX_TRACE_PACKET_EVENTS              0x00100000
NX_TRACE_RARP_EVENTS                0x00200000
NX_TRACE_TCP_EVENTS                 0x00400000
NX_TRACE_UDP_EVENTS                 0x00800000
UX_TRACE_ALL_EVENTS                 0x7F000000
UX_TRACE_ERRORS                     0x01000000
UX_TRACE_HOST_STACK_EVENTS          0x02000000
UX_TRACE_DEVICE_STACK_EVENTS        0x04000000
UX_TRACE_HOST_CONTROLLER_EVENTS     0x08000000
UX_TRACE_DEVICE_CONTROLLER_EVENTS   0x10000000
UX_TRACE_HOST_CLASS_EVENTS          0x20000000
UX_TRACE_DEVICE_CLASS_EVENTS        0x40000000
```

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

```
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.

> **IMPORTANT**
>
> 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.

> **IMPORTANT**
>
> 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.

> **IMPORTANT**
> 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.

> **IMPORTANT**
> 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

```
y_trace_is_full(void *trace_buffer_start)

{

     /* 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.

IMPORTANT

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 = tx_trace_user_event_insert (3000, 1, 2, 3, 4);

/* If status is TX_SUCCESS the user 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_isr_exit_insert, tx_trace_buffer_full_notify

Chapter 6 - Azure RTOS ThreadX trace events

7/20/2020 • 23 minutes to read

This chapter describes the Azure RTOS ThreadX events.

List of Events and Icons

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

ICON	MEANING
	Internal thread resume
	Internal thread suspend
	Interrupt Service Routine (ISR) Enter
	Interrupt Service Routine (ISR) Exit
	Internal time-slice
	Running
	**Block pool allocate** (*tx_block_allocate*)
	**Block pool create** (*tx_block_pool_create*)
	**Block pool delete** (*tx_block_pool_delete*)
	**Block pool information get** (*tx_block_pool_info_get*)
	**Block pool performance information get** (*tx_block_pool_performance_info_get*)
	**Block pool system performance information get** (*tx_block_pool_performance_system_info_get*)
	**Block pool prioritize** (*tx_block_pool_prioritize*)
	Block release to pool (*tx_block_release*)
	**Byte pool allocate memory** (*tx_byte_allocate*)
	**Byte pool create** (*tx_byte_pool_create*)

ICON	MEANING
	**Byte pool delete** (*tx_byte_pool_delete*)
	**Byte pool information get** (*tx_byte_pool_info_get*)
	**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*)

ICON	MEANING
	**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_semaphroe_performance_info_get*)

ICON	MEANING
PS	**Semaphore system performance information get** (*tx_semaphore_performance_system_info_get*)
SR	**Semaphore prioritize** (*tx_semaphore_prioritize*)
SP	**Semaphore put** (*tx_semaphore_put*)
SN	**Semaphore put notify** (*tx_semaphore_put_notify*)
TC	**Thread create** (*tx_thread_create*)
TD	**Thread delete** (*tx_thread_delete*)
TE	**Thread exit/entry notify** (*tx_thread_entry_exit_notify*)
TI	**Thread identify** (*tx_thread_identify*)
IG	**Thread information get** (*tx_thread_info_get*)
PI	**Thread performance information get** (*tx_thread_performance_info_get*)
PS	**Thread performance system information get** (*tx_thread_performance_system_info_get*)
CT	**Thread preemption change** (*tx_thread_preemption_change*)
CP	**Thread priority change** (*tx_thread_priority_change*)
RQ	**Thread relinquish** (*tx_thread_relinquish*)
RS	**Thread reset** (*tx_thread_reset*)
TR	**Thread resume** (*tx_thread_resume*)
TZ	**Thread Sleep** (*tx_thread_sleep*)*
SE	**Thread stack error notify** (*tx_thread_stack_error_notify*)
TS	**Thread suspend** (*tx_thread_suspend*)
TT	**Thread terminate** (*tx_thread_terminate*)
CS	**Thread time-slice change** (*tx_thread_time_slice_change*)

ICON	MEANING
	**Thread wait abort** (*tx_thread_wait_abort*)
	**Time get** (*tx_time_get*)
	**Time set** (*tx_time_set*)
	**Timer activate** (*tx_timer_activate*)
	**Timer change** (*tx_timer_change*)
	**Timer create** (*tx_timer_create*)
	**Timer deactivate** (*tx_timer_deactivate*)
	**Timer delete** (*tx_timer_delete*)
	**Timer information get** (*tx_timer_info_get*)
	**Timer performance information get** (*tx_timer_performance_info_get*)
	**Timer performance system information get** (*tx_timer_performance_system_info_get*)
	**User-Defined Event** (See Chapter 10)

Event Descriptions

Internal thread resume

Internal thread resume

Icon 

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:

THREAD STATE	VALUE
TX_READY	0

THREAD STATE	VALUE
TX_COMPLETED	1
TX_TERMINATED	2
TX_SUSPENDED	3
TX_SLEEP	4
TX_QUEUE_SUSP	5
TX_SEMAPHORE_SUSP	6
TX_EVENT_FLAG	7
TX_BLOCK_MEMORY	8
TX_BYTE_MEMORY	9
TX_TCP_IP	12
TX_MUTEX_SUSP	13

- Info Field 3: Stack pointer value during the call.
- Info Field 4: Pointer to next highest priority thread to execute.

Internal thread suspend

Internal thread suspend

Icon 

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:

THREAD STATE	VALUE
TX_COMPLETED	1
TX_TERMINATED	2
TX_SUSPENDED	3
TX_SLEEP	4
TX_QUEUE_SUSP	5

THREAD STATE	VALUE
TX_SEMAPHORE_SUSP	6
TX_EVENT_FLAG	7
TX_BLOCK_MEMORY	8
TX_BYTE_MEMORY	9
TX_TCP_IP	12
TX_MUTEX_SUSP	13

- 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

Enter ISR

Icon 

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

Exit ISR

Icon 

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.

Internal time-slice

Internal time-slice

Icon 

Description

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.

Running

Running in context

Icon 

Description

This event represents running within a thread context or idle system. It is used to illustrate subsequent changes in context as a result of an interrupt.

Information Fields

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

Block Allocate

tx_block_allocate

Icon 

Description

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.

Information Fields

- 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

tx_block_pool_create

Icon 

Description

This event represents creating a memory block pool via tx_block_pool_create.

Information Fields

- Info Field 1: Pointer to the corresponding 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

tx_block_pool_delete

Icon 

Description

This event represents deleting a memory block pool via tx_block_pool_delete.

Information Fields

- 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

tx_block_pool_info_get

Icon 

Description

This event represents getting information about a memory block pool via tx_block_pool_info_get.

Information Fields

- 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 Performance Information Get

tx_block_pool_performance_info_get

Icon 

Description

This event represents getting performance information about a memory block pool via tx_block_pool_performance_info_get.

Information Fields

- Info Field 1: Pointer to the block pool control block.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Block Pool Performance System Information Get

tx_block_pool_performance_system_info_get

Icon 

Description

This event represents getting performance information about all memory block pools via tx_block_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.

Block Pool Prioritize

tx_block_pool_prioritize

Icon 

Description

This event represents placing the highestpriority 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.

Information Fields

- Info Field 1: Memory block pool pointer.
- Info Field 2: Number of threads suspended on this block pool.
- Info Field 3: Stack pointer at the time of the call.
- Info Field 4: Not used.

Block Release

tx_block_release

Icon 

Description

This event represents releasing a previously allocated block back to the block pool.

Information Fields

- Info Field 1: Memory block pool pointer.
- Info Field 2: Pointer to block to release.
- Info Field 3: Number of threads suspended on this block pool.
- Info Field 4: Stack pointer at the time of the call.

Byte Allocate

tx_byte_allocate

Icon 

Description

This event represents allocating memory via tx_byte_allocate. If successful, the address of the memory allocated is returned in the second information field.

Information Fields

- Info Field 1: Pointer to the corresponding byte pool.
- Info Field 2: Pointer to the memory returned (if successful).
- Info Field 3: Number of bytes requested. Info Field 4: The wait option supplied to the tx_byte_allocate call.

Byte Pool Create

tx_byte_pool_create

Icon 

Description

This event represents creating a byte pool via tx_byte_pool_create.

Information Fields

- Info Field 1: Pointer to the corresponding byte pool.
- Info Field 2: Pointer to the start of the memory area. Info Field 3: Number of bytes in the byte pool.

- Info Field 4: The stack pointer at the time of the call.

Byte Pool Delete

tx_byte_pool_delete

Icon 

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

Icon 

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_info_get

Icon 

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

Icon 

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

Icon 

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

Icon 

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.

Event Flags Create

tx_event_flags_create

Icon 

Description

This event represents creating a new event flags group via tx_event_flags_create.

Information Fields

- Info Field 1: Pointer to event flags group control block.
- Info Field 2: Stack pointer at time of call.
- Info Field 3: Not used.
- Info Field 4: Not used.

Event Flags Delete

tx_event_flags_delete

Icon 

Description

This event represents deleting an event flags group via tx_event_flags_delete.

Information Fields

- Info Field 1: Pointer to event flags group.
- Info Field 2: Stack pointer at time of call.
- Info Field 3: Not used.
- Info Field 4: Not used.

Event Flags Get

tx_event_flags_get

Icon 

Description

This event represents retrieving event flags from an existing event flags group via tx_event_flags_get.

Information Fields

- Info Field 1: Pointer to event flags group.
- Info Field 2: Event flags requested.
- Info Field 3: Event flags currently set in the group.
- Info Field 4: Option requested on the event flags get.

Event Flags Information Get

tx_event_flags_info_get

Icon 

Description

This event represents retrieving information regarding an existing event flags group via tx_event_flags_info_get.

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.

Event Flags Performance Information Get

tx_event_flags_performance_info_get

Icon 

Description

This event represents retrieving performance information regarding an existing event flags group via tx_event_flags_performance_info_get.

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

Event Flags Performance System Info Get

tx_event_flags_performance_system_info_get

Icon 

Description

This event represents retrieving performance information regarding an existing event flags group via tx_event_flags_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.

Event Flags Set

tx_event_flags_set

Icon 

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

tx_event_flags_set_notify

Icon 

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

tx_interrupt_control

Icon 

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

tx_mutex_create

Icon 

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

tx_mutex_delete

Icon 

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

tx_mutex_get

Icon 

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

tx_mutex_info_get

Icon 

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

tx_mutex_performance_info_get

Icon 

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

tx_mutex_performance_system_info_get

Icon 

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

tx_mutex_prioritize

Icon 

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

tx_mutex_put

Icon 

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

tx_queue_create

Icon 

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.

Queue Delete

tx_queue_delete

Icon 

Description

This event represents deleting a queue via tx_queue_delete.

Information Fields

- Info Field 1: Pointer to queue.
- Info Field 2: Stack pointer at time of call.
- Info Field 3: Not used.
- Info Field 4: Not used.

Queue Flush

tx_queue_flush

Icon 

Description

This event represents flushing (clearing all queue contents) of a queue via tx_queue_flush.

Information Fields

- Info Field 1: Pointer to queue.
- Info Field 2: Stack pointer at time of call.
- Info Field 3: Not used.
- Info Field 4: Not used.

Queue Front Send

tx_queue_front_send

Icon 

Description

This event represents sending a message to the front of a queue via tx_queue_front_send.

Information Fields

- Info Field 1: Pointer to queue.
- Info Field 2: Pointer to start of message.
- Info Field 3: Wait option supplied to the
- tx_queue_front_send call.
- Info Field 4: Number of messages already enqueued.

Queue Information Get

tx_queue_info_get

Icon 

Description

This event represents getting information about a queue via tx_queue_info_get.

Information Fields

- Info Field 1: Pointer to queue.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Queue Performance Info Get

tx_queue_performance_info_get

Icon 

Description

This event represents getting performance information about a queue via tx_queue_performance_info_get.

Information Fields

- Info Field 1: Pointer to queue.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Queue Performance System Info Get

tx_queue_performance_system_info_get

Icon 

Description

This event represents getting system performance information about all the queues in the system.

Information Fields

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

Queue Prioritize

tx_queue_prioritize

Icon 

Description

This event represents prioritizing the queue's suspension list via tx_queue_prioritize.

Information Fields

- 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

tx_queue_receive

Icon 

Description

This event represents receiving a message from a queue via tx_queue_receive.

Information Fields

- 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

tx_queue_send

Icon 

Description

This event represents sending a message to a queue via tx_queue_send.

Information Fields

- 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

tx_queue_send_notify

Icon 

Description

This event represents registering a callback via tx_queue_send_notify which is called whenever a message is sent to a queue.

Information Fields

- Info Field 1: Pointer to queue.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Semaphore Ceiling Put

tx_semaphore_ceiling_put

Icon 

Description

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.

Information Fields

- 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

tx_semaphore_create

Icon 

Description

This event represents creating a semaphore via tx_semaphore_create.

Information Fields

- 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.

Semaphore Delete

tx_semaphore_delete

Icon 

Description

This event represents deleting a semaphore via tx_semaphore_delete.

Information Fields

- Info Field 1: Pointer to semaphore.
- nfo Field 2: Stack pointer at time of call.
- Info Field 3: Not used.
- Info Field 4: Not used.

Semaphore Get

tx_semaphore_get

Icon 

Description

This event represents obtaining a semaphore via tx_semaphore_get.

Information Fields

- 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.

Semaphore Information Get

tx_semaphore_info_get

Icon 

Description

This event represents obtaining information about a semaphore via tx_semaphore_info_get.

Information Fields

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

Semaphore Performance Info Get

tx_semaphore_performance_info_get

Icon 

Description

This event represents obtaining performance information about a semaphore via tx_semaphore_performance_info_get.

Information Fields

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

Semaphore Performance System Info

tx_semaphore_performance_system_info_get

Icon 

Description

This event represents obtaining performance information about all semaphores in the system via tx_semaphore_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.

Semaphore Prioritize

tx_semaphore_prioritize

Icon 

Description

This event represents prioritizing the semaphore's suspension list via tx_semaphore_prioritize.

Information Fields

- 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.

Semaphore Put

tx_semaphore_put

Icon 

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

tx_semaphore_put_notify

Icon 

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

tx_thread_create

Icon 

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.
- nfo Field 4: Size of stack in bytes.

Thread Delete

tx_thread_delete

Icon 

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

tx_thread_entry_exit_notify

Icon

Description

This event represents registering a callback via tx_thread_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

tx_thread_identify

Icon

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.

Thread Information Get

tx_thread_info_get

Icon

Description

This event represents getting information about the specified thread via tx_thread_info_get.

Information Fields

- 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

Icon

Description

This event represents getting performance information about the specified thread via

tx_thread_performance_info_get.

Information Fields

- 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

Icon 

Description

This event represents getting performance information about all threads via tx_thread_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.

Thread Preemption Change

tx_thread_preemption_change

Icon 

Description

This event represents changing a thread's preemption-threshold via tx_thread_preemption_change.

Information Fields

- 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

Icon 

Description

This event represents changing a thread's priority via tx_thread_priority_change.

- Information Fields
- 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

Icon 

Description

This event represents relinquishing the processor from a thread via tx_thread_relinquish.

Information Fields

- 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.

Thread Reset

tx_thread_reset

Icon 

Description

This event represents resetting a completed or terminated thread via tx_thread_reset.

Information Fields

- 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

Icon 

Description

This event represents resuming a suspended thread via tx_thread_resume.

Information Fields

- 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

Icon 

Description

This event represents suspending the current thread for a specified number of timer ticks via tx_thread_sleep.

Information Fields

- 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_event

Icon 

Description

This event represents registering a thread stack error notification routine via tx_thread_stack_error_notify_event.

Information Fields

- 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

Icon 

Description

This event represents suspending a thread via tx_thread_suspend.

Information Fields

- 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

Icon 

Description

This event represents terminating a thread via tx_thread_terminate.

Information Fields

- 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.
- Info Field 4: Not used.

Thread Time-Slice Change

tx_thread_time_slice_change

Icon 

Description

This event represents changing a thread's time-slice via tx_thread_time_slice_change.

Information Fields

- 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

tx_thread_wait_abort

Icon 

Description

This event represents aborting a thread's suspension via tx_thread_wait_abort.

Information Fields

- 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

tx_time_get

Icon 

Description

This event represents getting the current number of timer ticks via tx_time_get.

Information Fields

- 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

tx_time_set

Icon 

Description

This event represents setting the current number of timer ticks via tx_time_set.

Information Fields

- 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

tx_timer_activate

Icon 

Description

This event represents activating the specified timer via tx_timer_activate.

Information Fields

- Info Field 1: Pointer to timer.
- Info Field 2: Not used.
- Info Field 3: Not used.

- Info Field 4: Not used.

Timer Change

tx_timer_change

Icon 

Description

This event represents changing the specified timer via tx_timer_change.

Information Fields

- 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 

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 

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 

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 

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 

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 

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.

Chapter 7 - Azure RTOS FileX trace events

7/20/2020 • 17 minutes to read

This chapter contains a description of the Azure RTOS FileX events.

List of Events and Icons

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

The following describes each event:

ICON	MEANING
	Internal Logical Sector Cache Miss
	Internal Directory Cache Miss
	Internal Media Flush
	Internal Directory Entry Read
	Internal Directory Entry Write
	Internal I/O Driver Read
	Internal I/O Driver Write
	Internal I/O Driver Flush
	Internal I/O Driver Abort
	Internal I/O Driver Initialize
	Internal I/O Driver Boot Read
	Internal I/O Driver Release Sectors
	Internal I/O Driver Boot Write
	Internal I / O Driver Driver Un-initialize
	**Directory Attributes Read** (*fx_directory_attributes_read*)
	**Directory Attributes Set** (*fx_directory_attributes_set*)

ICON	MEANING
	**Directory Create** (*fx_directory_create*)
	**Directory Default Get** (*fx_directory_default_get*)
	**Directory Default Set** (*fx_directory_default_set*)
	**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*)

ICON	MEANING
	**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** (*fx_file_open*)
	**File Read** (*fx_file_read*)
	**File Relative Seek** (*fx_file_relative_seek*)
	**File Rename** (*fx_file_rename*)
	**File Seek** (*fx_file_seek*)
	**File Truncate** (*fx_file_truncate*)
	**File Truncate Release** (*fx_file_truncate_release*)
	**File Write** (*fx_file_write*)
	**Media Abort** (*fx_media_abort*)
	**Media Cache Invalidate** (*fx_media_cache_invalidate*)
	**Media Check** (*fx_media_check*)
	**Media Close** (*fx_media_close*)
	**Media Flush** (*fx_media_flush*)
	**Media Format** (*fx_media_format*)
	**Media Open** (*fx_media_open*)
	**Media Read** (*fx_media_read*)

ICON	MEANING
	**Media Space Available** (*fx_media_space_available*)
	**Media Volume Get** (*fx_media_volume_get*)
	**Media Volume Set** (*fx_media_volume_set*)
	**Media Write** (*fx_media_write*)
	**System Date Get** (*fx_system_date_get*)
	**System Date Set** (*fx_system_date_set*)
	**System Initialize** (*fx_system_initialize*)
	**System Time Get** (*fx_system_time_get*)
	**System Time Set** (*fx_system_time_set*)
	**Unicode Directory Create** (*fx_unicode_directory_create*)
	**Unicode Directory Rename** (*fx_unicode_directory_rename*)
	**Unicode File Create** (*fx_unicode_file_create*)
	**Unicode File Rename** (*fx_unicode_file_rename*)
	**Unicode Lenght Get** (*fx_unicode_length_get*)
	**Unicode Name Get** (*fx_unicode_name_get*)
	**Unicode Short Name Get** (*fx_unicode_short_name_get*)

Event Descriptions

The following describes each individual event.

Internal Logical Sector Cache Miss

Internal logical sector cache miss

Icon 

Description

This event represents an internal FileX logical sector cache miss.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Sector.
- Info Field 3: Total misses.
- Info Field 4: Cache size.

Internal Directory Cache Miss

Internal directory cache miss

Icon 

Description

This event represents an internal FileX directory cache miss.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Total misses.
- Info Field 3: Not used.
- Info Field 4: Not used.

Internal Media Flush

Internal media flush

Icon 

Description

This event represents an internal FileX media flush.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Number of dirty sectors.
- Info Field 3: Not used.
- Info Field 4: Not used.

Internal Directory Entry Read

Internal directory entry read

Icon 

Description

This event represents an internal FileX directory entry read 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.

Internal Directory Entry Write

Internal directory entry write

Icon 

Description

This event represents an internal FileX directory entry write 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.

Internal I/O Driver Read

Internal I/O driver read

Icon 

Description

This event represents an internal FileX I/O driver read event.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Sector.
- Info Field 3: Number of sectors.
- Info Field 4: Buffer pointer.

Internal I/O Driver Write

Internal I/O driver write

Icon 

Description

This event represents an internal FileX I/O driver write event.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Sector.
- Info Field 3: Number of sectors.
- Info Field 4: Buffer pointer.

Internal I/O Driver Flush

Internal I/O driver flush

Icon 

Description

This event represents an internal FileX I/O driver flush 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.

Internal I/O Driver Abort

Internal I/O dirver abort

Icon 

Description

This event represents an internal FileX I/O driver 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.

Internal I/O Driver Initialize

Internal I/O driver initialize

Icon 

Description

This event represents an internal FileX I/O driver initialize 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.

Internal I/O Driver Boot Sector Read

Internal I/O driver boot sector read

Icon 

Description

This event represents an internal FileX I/O driver boot sector read event.

Information Fields

- 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 Release Sectors

Internal I/O driver release sectors

Icon 

Description

This event represents an internal FileX I/O driver release sectors event.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Sector.
- Info Field 3: Number of sectors.
- Info Field 4: Not used.

Internal I/O Driver Boot Sector Write

Internal I/O driver boot sector write

Icon 

Description

This event represents an internal FileX I/O driver boot sector write event.

Information Fields

- 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 Un-initialize

Internal I/O driver un-initialize

Icon 

Description

This event represents an internal FileX I/O driver un-initialize 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.

Directory Attributes Read

fx_directory_attributes_read

Icon 

Description

This event represents a directory attributes read event.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to directory name.
- Info Field 3: Attributes bit map:

ATTRIBUTE	VALUE
Read Only	(0x01)
Hidden	(0x02)
System	(0x04)
Volume	(0x08)
Directory	(0x10)
Archive	(0x20)

- Info Field 4: Not used.

Directory Attributes Set

fx_directory_attributes_set

Icon 

Description

This event represents a directory a directory attributes set event.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to directory name.
- Info Field 3: Attributes bit map:

ATTRIBUTE	VALUE
Read Only	(0x01)
Hidden	(0x02)
System	(0x04)
Archive	(0x20)

- Info Field 4: Not used.

Directory Create

fx_directory_create

Icon 

Description

This event represents a directory create 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 Default Get

fx_directory_default_get

Icon 

Description

This event represents a directory default set 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 Default Set

fx_directory_default_set

Icon 

Description

This event represents a directory default set event.

Information Fields

- 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

Icon 

Description

This event represents a directory delete 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 First Entry Find

fx_directory_first_entry_find

Icon 

Description

This event represents a directory first 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 First Full Entry Find

fx_directory_first_full_entry_find

Icon 

Description

This event represents a directory first full 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 Information Get

fx_directory_information_get

Icon 

Description

This event represents a directory information get 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 Local Path Clear

fx_directory_local_path_clear

Icon 

Description

This event represents a directory local path clear 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.

Directory Local Path Get

fx_directory_local_path_get

Icon 

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

fx_directory_local_path_restore

Icon 

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

fx_directory_local_path_set

Icon 

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

fx_directory_long_name_get

Icon 

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

fx_directory_name_test

Icon 

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

fx_directory_next_entry_find

Icon 

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

fx_directory_next_full_entry_find

Icon 

Description

This event represents a directory next full 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 Rename

fx_directory_rename event

Icon 

Description

This event represents a directory rename event.

Information Fields

- 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

fx_directory_short_name_get

Icon 

Description

This event represents a directory short name get event.

Information Fields

- 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

fx_file_allocate

Icon 

Description

This event represents a file allocate event.

Information Fields

- Info Field 1: Pointer to the file.
- Info Field 2: Requested size.
- Info Field 3: Current size.
- Info Field 4: New size.

File Attributes Read

fx_file_attributes_read

Icon 

Description

This event represents a file attributes read event.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Attributes bit map:

ATTRIBUT	VALUE
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

fx_file_attributes_set

Icon 

Description

This event represents a file attributes set event.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to file name.
- Info Field 3: Attributes bit map:

ATTRIBUTE	VALUE
Read Only	(0x01)

ATTRIBUTE	VALUE
Hidden	(0x02)
System	(0x04)
Archive	(0x20)

- Info Field 4: Not used.

File Best Effort Allocate

fx_file_best_effort_allocate

Icon 

Description

This event represents a file best effort allocate event.

Information Fields

- 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

fx_file_close

Icon 

Description

This event represents a file close event.

Information Fields

- Info Field 1: Pointer to the file.
- Info Field 2: File size.
- Info Field 3: Not used.
- Info Field 4: Not used.

File Create

fx_file_create

Icon 

Description

This event represents a file create event.

Information Fields

- 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

fx_file_date_time_set

Icon 

Description

This event represents a file date/time set event.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to file name.
- Info Field 3: Year.
- Info Field 4: Month.

File Delete

fx_file_delete

Icon 

Description

This event represents a file delete event.

Information Fields

- 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 Open

fx_file_open

Icon 

Description

This event represents a file open event.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to the file control block.
- Info Field 3: Pointer to file name.
- Info Field 4: Open type:

OPEN TYPE	VALUE
Open for Read	(0x00)
Open for Write	(0x01)
Fast Open for Read	(0x02)

File Read

fx_file_read

Icon 

Description

This event represents a file read 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 read.

File Relative Seek

fx_file_relative_seek

Icon 

Description

This event represents a file relative seek event.

Information Fields

- Info Field 1: Pointer to the file.
- Info Field 2: Byte offset.
- Info Field 3: Seek from:

EVENT	VALUE
From Beginning	(0x00)
From End	(0x01)
Forward	(0x02)
Backward	(0x03)

- Info Field 4: Previous offset.

File Rename

fx_file_rename

Icon 

Description

This event represents a file rename event.

Information Fields

- Info Field 1: Pointer to the media.
- Info Field 2: Pointer to old file name.
- Info Field 3: Pointer to new file name.
- Info Field 4: Not used.

File Seek

fx_file_seek

Icon 

Description

This event represents a file seek event.

Information Fields

- Info Field 1: Pointer to the file.
- Info Field 2: Byte offset.
- Info Field 3: Previous offset.
- Info Field 4: Not used.

File Truncate

fx_file_truncate

Icon 

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

fx_file_truncate_release

Icon 

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

fx_file_write

Icon 

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

fx_media_abort

Icon 

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

fx_media_cache_invalidate

Icon 

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

fx_media_check

Icon 

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:

ERROR TYPE	VALUE
FAT Chain Error	(0x01)
Directory Error	(0x02)
Lost Cluster Error	(0x04)
File Size Error	(0x08)

Media Close

fx_media_close

Icon 

Description

This event represents a media close 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 Flush

fx_media_flush

Icon 

Description

This event represents a media flush 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 Format

fx_media_format

Icon 

Description

This event represents a media format event.

Information Fields

- 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

fx_media_open

Icon 

Description

This event represents a media open event.

Information Fields

- 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 Media Read

fx_media_read

Icon 

Description

This event represents a media read event.

Information Fields

- 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

fx_media_space_available

Icon 

Description

This event represents a media space available event.

Information Fields

- 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

fx_media_volume_get

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

fx_media_volume_set

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

fx_media_write

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

fx_system_date_get

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

fx_system_date_set

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

fx_system_initialize

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.

System Time Get

fx_system_time_get

Icon 

Description

This event represents a system time get event.

Information Fields

- Info Field 1: Hour.
- Info Field 2: Minute.
- Info Field 3: Second.
- Info Field 4: Not used.

System Time Set

fx_system_time_set

Icon 

Description

This event represents a system time set event.

Information Fields

- Info Field 1: Hour.
- Info Field 2: Minute.
- Info Field 3: Second.
- Info Field 4: Not used.

Unicode Directory Create

fx_unicode_directory_create

Icon 

Description

This event represents a Unicode directory create event.

Information Fields

- 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.

Unicode Directory Rename

fx_unicode_directory_rename

Icon 

Description

This event represents a Unicode directory rename event.

Information Fields

- 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.

Unicode File Create

fx_unicode_file_create

Icon 

Description

This event represents a Unicode file create event.

Information Fields

- 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.

Unicode File Rename

fx_unicode_file_rename

Icon 

Description

This event represents a Unicode file rename event.

Information Fields

- 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.

Unicode Length Get

fx_unicode_length_get

Icon 

Description

This event represents a Unicode length get event.

Information Fields

- 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 

Description

This event represents a Unicode name get event.

Information Fields

- 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 

Description

This event represents a Unicode short name get event.

Information Fields

- 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.

Chapter 8 - Azure RTOS NetX trace events

7/20/2020 • 42 minutes to read

This chapter contains a description of the Azure RTOS NetX events.

List of Events and Icons

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

ICON	MEANING
	Internal ARP Request Receive
	Internal ARP Request Send
	Internal ARP Response Receive
	Internal ARP Response Send
	Internal ICMP Receive
	Internal ICMP Send
	Internal NetX IGMP Receive
	Internal IP Receive
	Internal IP Send
	Internal TCP Data Receive
	Internal TCP Data Send
	Internal TCP FIN Receive
	Internal TCP FIN Send
	Internal TCP RST Receive
	Internal TCP RST Send
	Internal TCP SYN Receive

ICON	MEANING
	Internal TCP SYN Send
	Internal UDP Receive
	Internal UDP Send
	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

ICON	MEANING
	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 Entry Create** (*nx_arp_static_entry_create*)
	**ARP Static Entries Delete** (*nx_arp_static_entries_delete*)
	**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*)

ICON	MEANING
	**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*)

ICON	MEANING
	**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*)

ICON	MEANING
	**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 Static Route Add** (*nx_ip_static_route_add*)
	**IP Static Route Delete** (*nx_ip_static_route_delete)*
	**IP Status Check** (*nx_ip_status_check*)
	**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*)

ICON	MEANING
	**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*)

ICON	MEANING
	**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*)

ICON	MEANING
	**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 Socket Unbind** (*nx_udp_socket_unbind*)
	**UDP Source Extract** (*nx_udp_source_extract*)

Event Descriptions

The following pages describe the NetX Trace Events.

Internal ARP Request Receive

Internal ARP request receive

Icon 

Description

This event represents an internal NetX ARP request 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: Not used

Internal ARP Request Send

Internal ARP request send

Icon 

Description

This event represents an internal NetX ARP request 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: Not used

Internal ARP Response Receive

Internal ARP request receive

Icon 

Description

This event represents an internal NetX ARP response 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: Not used

Internal ARP Response Send

Internal ARP request send

Icon 

Description

This event represents an internal NetX response 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: Not used

Internal ICMP Receive

Internal ICMP receive

Icon 

Description

This event represents an internal NetX ICMP 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: Word 0 of ICMP header

Internal ICMP Send

Internal ICMP send

Icon 

Description

This event represents an internal NetX ICMP 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: Word 0 of ICMP header

Internal IGMP Receive

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

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

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

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

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

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

Internal TCP FIN Send

Internal TCP fin send

Icon 

Description

This event represents an internal NetX TCP FIN 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 RST Receive

Internal TCP RST receive

Icon 

Description

This event represents an internal NetX TCP reset 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.

Internal TCP RST Send

Internal TCP RST send

Icon 

Description

This event represents an internal NetX TCP reset 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 SYN Receive

Internal TCP SYN receive

Icon 

Description

This event represents an internal NetX TCP SYN 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

Internal TCP SYN Send

Internal TCP SYN send

Icon 

Description

This event represents an internal NetX TCP SYN 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 UDP Receive

Internal UDP receive

Icon 

Description

This event represents an internal NetX UDP 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: Word 0 of UDP header

Internal UDP Send

Internal UDP send

Icon 

Description

This event represents an internal NetX UDP 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: Word 0 of UDP header

Internal RARP Receive

Internal RARP receive

Icon 

Description

This event represents an internal NetX RARP receive event.

Information Fields

- 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

Internal RARP send

Icon 

Description

This event represents an internal NetX RARP send event.

Information Fields

- 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

Internal TCP retry

Icon 

Description

This event represents an internal NetX TCP retry 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: Number of retries

Internal TCP State Change

Internal TCP state change

Icon 

Description

This event represents an internal NetX TCP socket state change event.

Information Fields

- 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

Internal I/O driver packet send

Icon 

Description

This event represents an internal NetX I/O driver packet 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 Initialize

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

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

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

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

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

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

Internal I/O Driver RARP Send

Internal I/O driver RARP send

Icon 

Description

This event represents an internal NetX I/O driver RARP 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 Multicast Join

Internal I/O driver multicast join

Icon 

Description

This event represents an internal NetX I/O driver multicast join 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 Multicast Leave

Internal I/O driver multicast leave

Icon 

Description

This event represents an internal NetX I/O driver multicast leave 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 Status

Internal I/O driver get status

Icon 

Description

This event represents an internal NetX I/O driver get status 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 Speed

Internal I/O driver get speed

Icon 

Description

This event represents an internal NetX I/O driver get speed 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 Duplex Type

Internal I/O driver get duplex type

Icon 

Description

This event represents an internal NetX I/O driver get duplex type 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 Error Count

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

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

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

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

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

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

nx_arp_dynamic_entries_invalidate

Icon 

Description

This event represents invalidating all dynamic ARP entires 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

nx_arp_dynamic_entry_set

Icon 

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

nx_arp_enable

Icon 

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

nx_arp_gratuitous_send

Icon 

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

nx_arp_hardware_address_find

Icon 

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

nx_arp_info_get

Icon 

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

nx_arp_ip_address_find

Icon 

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 Entry Create

nx_arp_static_entry_create

Icon 

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 Entries Delete

nx_arp_static_entries_delete

Icon 

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 Delete

nx_arp_static_entry_delete

Icon 

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

nxd_und_cache_entry_delete

Icon 

Description

This event represents deleting an entry in the neighbor cache table via nx_udp_socket_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

nxd_nd_cache_entry_set

Icon 

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

Duo Cache Invalidate

nxd_nd_cache_invalidate

Icon 

Description

This event represents invalidating the entire neighbor cache table via nxd_nd_cache_invalidate.

Information Fields

- 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

nxd_nd_cache_ip_address_find

Icon 

Description

This event represents retrieving an IP address matching the supplied physical address from the cache table via nxd_nd_cache_ip_address_find.

Information Fields

- 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

nxd_icmp_enable

Icon 

Description

This event represents ICMPv4 and ICMPv6 services being enabled on the specified IP instance via nxd_icmp_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

Duo ICMP Ping

nxd_icmp_ping

Icon 

Description

This event represents sending a ping (echo request) to an IPv6 host via nxd_icmp_ping.

Information Fields

- 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

nxd_ip_max_payload_size

Icon 

Description

This event computes the max payload the specified packet can carry without requiring fragmentation.

Information Fields

- 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

nxd_ip_max_packet_send

Icon 

Description

This event represents sending a raw IP packet out the specified network interface to the supplied IP destination addressvia nxd_ip_raw_packet_send.

Information Fields

- 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

Duo IPv6 Default Router Add

nxd_ipv6_default_router_add

Icon 

Description

This event represents adding a default router to the IP instance's IPv6 routing table via nxd_ipv6_default_router_add.

Information Fields

- Info Field 1: Pointer to IP instance.
- Info Field 2: Destination Network address.
- Info Field 3: Life time information.
- Info Field 4: Not used.

Duo IPv6 Default Router Delete

nxd_ipv6_default_router_delete

Icon 

Description

This event represents removing a default router from the IP instance's IPv6 routing table via nxd_ipv6_default_router_delete.

Information Fields

- Info Field 1: Pointer to IP instance.
- Info Field 2: Fourth word (least significant) of the default router IPv6 address.
- Info Field 3: Not used.
- Info Field 4: Not used.

Duo IPv6 Enable

nxd_ipv6_enable

Icon 

Description

This event represents enabling IPv6 services on the supplied IP instance via nxd_ipv6_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

Duo IPv6 Global Address Get

nxd_ipv6_global_address_get

Icon 

Description

This event represents retrieving the global (primary) IP address on the IP instance located at index 1 in the IP instance interface table via nxd_ipv6_global_address_get.

Information Fields

- Info Field 1: Pointer to IP instance.
- Info Field 2: Fourth word (least significant) of the global address
- Info Field 3: IPv6 address prefix length.
- Info Field 4: Index into IP interface table (1).

Duo IPv6 Global Address Set

nxd_ipv6_global_address_set

Icon 

Description

This event represents setting the global (primary) IP address on the IP instance located at index 1 in the IP instance interface table via nxd_ipv6_global_address_set.

Information Fields

- Info Field 1: Pointer to IP instance
- Info Field 2: Fourth word (least significant) of the global address
- Info Field 3: IPv6 address prefix length
- Info Field 4: Index into IP interface table (1)

Duo IPv6 Initiate Dad Process

nxd_ipv6_initiate_dad_process

Icon 

Description

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 nxd_ipv6_initiate_dad_process.

Information Fields

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

Duo IPv6 Interface Address Get

nxd_ipv6_interface_address_get

Icon 

Description

This event represents retrieving the IP address and prefix at the specified index into the IP instance interface address table via nxd_ipv6_interface_address_get.

Information Fields

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

Duo IPv6 Interface Address Set

nxd_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

nxd_ipv6_linklocal_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 IP v6 link local address
- Info Field 3: Not used
- Info Field 4: Not used

Duo IPv6 Link Local Address Set

nxd_ipv6_linklocal_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 IPv6 Raw Packet Send

nxd_ipv6_raw_packet_send

Icon 

Description

This event represents sending a raw IP packet through the primary IP interface to the speficied destination via nxd_ip_raw_packet_send.

Information Fields

- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to packet to send
- Info Field 3: Destination IP address
- Info Field 4: Not used

Duo TCP Socket Peer Info Get

nxd_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: The lease significant 32-bit of the IP address

Duo TCP Socket Set Interface

nxd_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 Send

nxd_udp_socket_send

Icon 

Description

This event represents sending a UDP packet through the specified socket with the input IP address and port via nxd_udp_socket_send.

Information Fields

- Info Field 1: Pointer UDP Socket
- Info Field 2: Pointer to UDP packet
- Info Field 3: Packet length
- Info Field 4: Not used

Duo UDP Socket Set Interface

nxd_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

Duo UDP Source Extract

nxd_udp_socket_extract

Icon 

Description

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.

Information Fields

- 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

ICMP Enable

nx_icmp_enable

Icon 

Description

This event represents enabling ICMP via nx_icmp_enable.

Information Fields

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

ICMP Information Get

nx_icmp_info_get

Icon 

Description

This event represents getting information via nx_icmp_info_get.

Information Fields

- Info Field 1: Pointer to the IP instance
- Info Field 2: Pings sent
- Info Field 3: Ping responses
- Info Field 4: Pings received

ICMP Ping

nx_icmp_ping

Icon 

Description

This event represents pinging a target IP address via nx_icmp_ping.

Information Fields

- 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

IGMP Enable

nx_icmp_enable

Icon 

Description

This event represents enabling IGMP via nx_igmp_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

IGMP Information Get

nx_icmp_info_get

Icon 

Description

This event represents getting information via nx_igmp_info_get.

Information Fields

- Info Field 1: Pointer to the IP instance
- Info Field 2: Reports sent
- Info Field 3: Queries received
- Info Field 4: Groups joined

IGMP Loopback Disable

nx_igmp_loopback_disable

Icon 

Description

This event represents disabling IGMP loopback via nx_igmp_loopback_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

IGMP Loopback Enable

nx_igmp_loopback_enable

Icon 

Description

This event represents enabling IGMP loopback via nx_igmp_loopback_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

IGMP Multicast Join

nx_igmp_multicast_join

Icon 

Description

This event represents joining a multicast group via nx_igmp_multicast_join.

Information Fields

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

IGMP Multicast Leave

nx_igmp_multicast_leave

Icon 

Description

This event represents leaving a multicast group via nx_igmp_multicast_leave.

Information Fields

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

IP Address Change Notify

nx_ip_address_change_notify

Icon 

Description

This event represents registering for IP change notification via nx_ip_address_change_notify.

Information Fields

- Info Field 1: Pointer to the IP instance
- Info Field 2: Callback function pointer
- Info Field 3: Additional information pointer
- Info Field 4: Not used

IP Address Get

nx_ip_address_get

Icon 

Description

This event represents getting the IP address via nx_ip_address_get.

Information Fields

- 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

Icon 

Description

This event represents setting the IP address via nx_ip_address_set.

Information Fields

- 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

Icon 

Description

This event represents creating an IP instance via nx_ip_create.

Information Fields

- 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

Icon 

Description

This event represents deleting an IP instance via nx_ip_delete.

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 Driver Direct Command

nx_ip_driver_direct_command

Icon 

Description

This event represents a direct I/O driver command via nx_ip_driver_direct_command.

Information Fields

- 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

Icon 

Description

This event represents disabling IP forwarding via nx_ip_forwarding_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 Forwarding Enable

nx_ip_forwarding_enable

Icon 

Description

This event represents enabling IP forwarding via nx_ip_forwarding_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 Fragment Disable

nx_ip_fragment_disable

Icon 

Description

This event represents disabling IP fragmenting via nx_ip_fragment_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 Fragment Enable

nx_ip_fragment_enable

Icon 

Description

This event represents enabling IP fragmenting via nx_ip_fragment_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 Gateway Address Set

nx_ip_gateway_address_set

Icon 

Description

This event represents setting the gateway IP address via nx_ip_gateway_address_set.

Information Fields

- 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

Icon 

Description This event represents getting IP information via nx_ip_info_get.

Information Fields

- 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_interface_attach

Icon 

Description

This event represents a secondary network interface being attached to the IP instance via nx_ip_interface_attach.

Information Fields

- 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

nx_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

nx_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

nx_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

nx_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

nx_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

nx_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

IP Static Route Delete

nx_ip_static_route_delete

Icon 

Description

This event represents a static route being removed from the IP instance routing table via nx_ip_static_route_delete.

Information Fields

- 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

Icon 

Description

This event represents checking for an IP status via nx_ip_status_check.

Information Fields

- 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

Icon 

Description

This event represents enabling IPSec services on the supplied IP instance via nx_ipsec_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

Packet Allocate

nx_packet_allocate

Icon 

Description

This event represents allocating a packet via nx_packet_allocate.

Information Fields

- 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

Icon 

Description

This event represents copying a packet via nx_packet_copy.

Information Fields

- 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

Icon 

Description

This event represents appending data to a packet via nx_packet_data_append.

Information Fields

- 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

Packet Data Extract Offset

nx_udp_source_extract_offset

Icon 

Description

This event represents packet data that is extracted into a supplied buffer from a packet via nx_udp_source_extract_offset.

Information Fields

- 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 

Description

This event represents retrieving data from a packet via nx_packet_data_retrieve.

Information Fields

- 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 

Description

This event represents getting the length of a packet via nx_packet_length_get.

Information Fields

- 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 

Description

This event represents creating a packet pool via nx_packet_pool_create.

Information Fields

- 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 

Description

This event represents deleting a packet pool via nx_packet_pool_delete.

Information Fields

- 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 

Description

This event represents getting packet pool information via nx_packet_pool_info_get.

Information Fields

- Info Field 1: Pointer to packet pool
- Info Field 2: Total packets
- Info Field 3: Available packets
- Info Field 4: Empty requests

Packet Release

nx_packet_data_release

Icon 

Description

This event represents releasing a packet via nx_packet_release.

Information Fields

- Info Field 1: Pointer to the packet
- Info Field 2: Packet status
- Info Field 3: Available packets
- Info Field 4: Not used

Packet Transmit Release

nx_packet_transmit_release

Icon 

Description

This event represents releasing a transmit packet via nx_packet_transmit_release.

Information Fields

- Info Field 1: Pointer to the packet

- Info Field 2: Packet status
- Info Field 3: Available packets
- Info Field 4: Not used

RARP Disable

nx_rarp_disable

Icon 

Description

This event represents disabling RARP via nx_rarp_disable.

Information Fields

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

RARP Enable

nx_rarp_enable

Icon 

Description

This event represents enabling RARP via nx_rarp_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

RARP Information Get

nx_rarp_info_get

Icon 

Description

This event represents getting RARP information via nx_rarp_info_get.

Information Fields

- Info Field 1: Pointer to IP instance
- Info Field 2: Requests sent
- Info Field 3: Responses received
- Info Field 4: Invalid responses

System Initialize

nx_system_initialize

Icon 

Description

This event represents initializing NetX via nx_system_initialize.

Information Fields

- Info Field 1: Not used
- Info Field 2: Not used
- Info Field 3: Not used
- Info Field 4: Not used

TCP Client Socket Bind

nx_tcp_client_socket_bind

Icon 

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

nx_tcp_client_socket_connect

Icon 

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

nx_tcp_client_socket_port_get

Icon 

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

nx_tcp_client_socket_unbind

Icon 

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

nx_tcp_enable

Icon 

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 TCP Free Port Find

nx_tcp_free_port_find

Icon 

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 Infomation Get

nx_tcp_info_get

Icon 

Description

This event represents retrieving TCP information for the specified IP instance via nx_tcp_info_get.

Information Fields

- Info Field 1: Pointer to IP instance
- Info Field 2: Number of bytes sent
- Info Field 3: Number of bytes received
- Info Field 4: Number of invalid packets

TCP Server Socket Accept

nx_tcp_server_socket_accept

Icon 

Description

This event represents setting up the server socket after an active connection request was received via nx_tcp_server_socket_accept.

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 Server Socket Listen

nx_tcp_server_socket_listen

Icon 

Description

This event represents register a listen request and a server socket for the specified TCP port via nx_tcp_server_socket_listen.

Information Fields

- Info Field 1: Pointer to IP instance
- Info Field 2: TCP port number
- Info Field 3: Pointer to socket
- Info Field 4: Maximum number of connections that can be queued

TCP Server Socket Relisten

nx_tcp_server_socket_relisten

Icon 

Description

This event represents register another server socket for an existing listen request on the specified TCP port via nx_tcp_server_socket_relisten.

Information Fields

- Info Field 1: Pointer to IP instance
- Info Field 2: TCP port number
- Info Field 3: Pointer to socket
- Info Field 4: Socket state

TCP Server Socket Unaccept

nx_tcp_server_socket_unaccept

Icon 

Description

This event represents removing the server socket from association with the port receiving an earlier passive connection via nx_tcp_server_socket_unaccept.

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 Server Socket Unlisten TCP Server Socket Unlisten

nx_tcp_server_socket_unlisten

Icon 

Description

This event represents removing a previous listen request for the specified TCP port via nx_tcp_server_socket_unlisten.

Information Fields

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

TCP Socket Bytes Available

nx_tcp_socket_bytes_available

Icon 

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

nx_tcp_socket_create

Icon 

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

nx_tcp_socket_delete

Icon 

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

nx_tcp_socket_disconnect

Icon 

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

nx_tcp_socket_info_get

Icon 

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

nx_tcp_socket_mss_get

Icon 

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

TCP Socket MSS Peer Get

nx_tcp_socket_mss_peer_get

Icon 

Description

This event represents getting the MSS value of the socket's peer via nx_tcp_socket_mss_peer_get.

Information Fields

- 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

nx_tcp_socket_mss_set

Icon 

Description

This event represents setting a socket's MSS via nx_tcp_socket_mss_set.

Information Fields

- 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

nx_tcp_socket_peer_info_get

Icon 

Description

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.

Information Fields

- 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

nx_tcp_socket_receive

Icon 

Description

This event represents receiving data from a socket via nx_tcp_socket_receive.

Information Fields

- 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 Receive Notify

nx_tcp_socket_receive_notify

Icon 

Description

This event represents registering a receive notify callback for a socket via nx_tcp_socket_receive_notify.

Information Fields

- 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 Send

nx_tcp_socket_send

Icon 

Description

This event represents sending data on a socket via nx_tcp_socket_send.

Information Fields

- Info Field 1: Pointer to socket
- Info Field 2: Pointer to packet
- Info Field 3: Length of packet
- Info Field 4: Transmit sequence number

TCP Socket State Wait

nx_tcp_socket_state_wait

Icon 

Description

This event represents waiting for a socket to enter a particular state via nx_tcp_socket_state_wait.

Information Fields

- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Desired socket state
- Info Field 4: Previous socket state

TCP Socket Transmit Configure

nx_tcp_socket_transmit_configure

Icon 

Description

This event represents configuring the transmit options for a socket via nx_tcp_socket_transmit_configure.

Information Fields

- Info Field 1: Pointer to IP instance
- Info Field 2: Pointer to socket
- Info Field 3: Transmit queue depth
- Info Field 4: Timeout value

TCP Socket Window Update Notify Set

nx_tcp_window_update_notify_set

Icon 

Description

This event represents a TCP socket receiving notification of an increase in the remote host receive window via nx_tcp_window_update_notify_set.

Information Fields

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

UDP Enable

nx_udp_enable

Icon 

Description

This event represents enabling UDP via nx_udp_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

UDP Free Port Find

nx_udp_free_port_find

Icon 

Description

This event represents finding a free UDP port via nx_udp_free_port_find.

Information Fields

- 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

nx_udp_info_get

Icon 

Description

This event represents getting information via nx_udp_info_get.

Information Fields

- 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

Icon 

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

Icon 

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

Icon 

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

Icon 

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

Icon 

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

Icon 

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

UDP Socket Information Get Event

nx_udp_socket_info_get event

Icon 

Description

This event represents getting information about a UDP socket via nx_udp_socket_info_get.

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

UDP Socket Interface Set

nx_udp_socket_interface_set event

Icon 

Description

This event represents setting the outgoing interface of the specified UDP socket with the specified interface via nx_udp_socket_interface_set.

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

UDP Socket Port Get

nx_udp_socket_port_get

Icon 

Description

This event represents retrieving the UDP port the specified UDP socket is bound to via nx_udp_socket_port_get.

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

UDP Socket Receive

nx_udp_socket_receive

Icon 

Description

This event represents receiving data on the specified UDP socket via nx_udp_socket_receive.

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

UDP Socket Receive Notify

nx_udp_socket_receive_notify

Icon 

s Description

This event represents registering a receive notify callback via nx_udp_socket_receive_notify.

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

UDP Socket Send

nx_udp_socket_send

Icon 

Description

This event represents sending data through a UDP socket via nx_udp_socket_send.

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

UDP Socket Unbind

nx_udp_socket_unbind

Icon 

Description

This event represents unbinding a UDP port with a socket via nx_udp_socket_unbind.

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

UDP Source Extract

nx_udp_socket_create

Icon 

Description

This event represents getting the IP address and port number of a received UDP packet via nx_udp_source_extract.

Information Fields

- Info Field 1: Pointer to packet
- Info Field 2: Sender's IP address
- Info Field 3: Sender's port number
- Info Field 4: Not used

Chapter 9 - Azure RTOS USBX trace events

7/20/2020 • 50 minutes to read

This chapter contains a description of the Azure RTOS USBX events displayed by TraceX.

List of Events and Icons

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

ICON	MEANING
	**Device Class Cdc Activate** *(ux_device_class_cdc_activate)*
	**Device Class Cdc Deactivate** *(ux_device_class_cdc_deactivate)*
	**Device Class Cdc Read** *(ux_device_class_cdc_read)*
	**Device Class Cdc Write** *(ux_device_class_cdc_write)*
	**Device Class Dpump Activate** *(ux_device_class_dpump_activate)*
	**Device Class Dpump Deactivate** *(ux_device_class_dpump_deactivate)*
	**Device Class Dpump Read** *(ux_device_class_dpump_read)*
	**Device Class Dpump Write** *(ux_device_class_dpump_write)*
	**Device Class Hid Activate** *(ux_device_class_hid_activate)*
	**Device Class Hid Deactivate** *(ux_device_class_hid_deactivate)*
	**Device Class Hid Descriptor Send** *(ux_device_class_hid_descriptor_send)*
	**Device Class Hid Event Get** *(ux_device_class_hid_event_get)*
	**Device Class Hid Event Set** *(ux_device_class_hid_event_set)*
	**Device Class Hid Report Get** *(ux_device_class_hid_report_get)*
	**Device Class Hid Report Set** *(ux_device_class_hid_report_set)*

ICON	MEANING
	Device Class Pima Activate *(ux_device_class_pima_activate)*
	Device Class Pima Deactivate *(ux_device_class_pima_deactivate)*
	Device Class Pima Device Info Send *(ux_device_class_pima_device_info_send)*
	Device Class Pima Event Get *(ux_device_class_pima_event_get)*
	Device Class Pima Event Set *(ux_device_class_pima_event_set)*
	Device Class Pima Object Add *(ux_device_class_pima_object_add)*
	Device Class Pima Object Data Get *(ux_device_class_pima_object_data_get)*
	Device Class Pima Object Data Send *(ux_device_class_pima_object_data_send)*
	Device Class Pima Object Delete *(ux_device_class_pima_object_delete)*
	Device Class Pima Object Handles Send *(ux_device_class_pima_object_handles_send)*
	Device Class Pima Object Info Get *(ux_device_class_pima_object_info_get)*
	Device Class Pima Object Info Send *(ux_device_class_pima_object_info_send)*
	Device Class Pima Objects Number Send *(ux_device_class_pima_objects_number_send)*
	Device Class Pima Partial Object Data Get *(ux_device_class_pima_partial_object_data_get)*
	Device Class Pima Response Send *(ux_device_class_pima_response_send)*
	Device Class Pima Storage Id Send *(ux_device_class_pima_storage_id_send)*
	Device Class Pima Storage Info Send *(ux_device_class_pima_storage_info_send)*
	Device Class Rndis Activate *(ux_device_class_rndis_activate)*

ICON	MEANING
	**Device Class Rndis Deactivate** *(ux_device_class_rndis_deactivate)*
	**Device Class Rndis Message Keep Alive** *(ux_device_class_rndis_msg_keep_alive)*
	**Device Class Rndis Message Query** *(ux_device_class_rndis_msg_query)*
	**Device Class Rndis Message Reset** *(ux_device_class_rndis_msg_reset)*
	**Device Class Rndis Message Set** *(ux_device_class_rndis_msg_set)*
	**Device Class Rndis Packet Receive** *(ux_device_class_rndis_packet_receive)*
	**Device Class Rndis Packet Transmit** *(ux_device_class_rndis_packet_transmit)*
	**Device Class Storage Activate** *(ux_device_class_storage_activate)*
	**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)*

ICON	MEANING
	**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)*

ICON	MEANING
	**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)*

ICON	MEANING
	**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)*

ICON	MEANING
	**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)*

ICON	MEANING
	**Host Class Hub Port Change Suspend Process** *(ux_host_class_hub_port_change_suspend_process)*
	**Host Class Pima Activate** *(ux_host_class_prima_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)*

ICON	MEANING
	**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)*

ICON	MEANING
	**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_prolific_reception_stop)*
	**Host Class Prolific Write** *(ux_host_class_prolific_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)*

ICON	MEANING
	**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)

ICON	MEANING
	**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)*
	**Host Stack Transfer Request Abort** *(ux_host_stack_transfer_request_abort)*
	**USBX Error** *(ux_error)*

Event Descriptions

The following pages describe the USBX Trace Events.

Device Class Cdc Activate

ux_device_class_cdc_activate

Icon 

Description

This event represents a USBX Device Class Cdc Activate Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Cdc Deactivate

ux_device_class_cdc_deactivate

Icon 

Description

This event represents a USBX Device Class Cdc Deactivate.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Cdc Read

ux_device_class_cdc_read

Icon 

Description

This event represents a USBX Device Class Cdc Read Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Data pointer.
- Info Field 3: Requested length.
- Info Field 4: Not used.

Device Class Cdc Write

ux_device_class_cdc_write

Icon 

Description

This event represents a USBX Device Class Cdc Write Event.

Information Fields

- Info Field 1: Class Instance.

- Info Field 2: Data pointer.
- Info Field 3: Requested length.
- Info Field 4: Not used.

Device Class Dpump Activate

ux_device_class_dpump_activate

Icon 

Description

This event represents a USBX Device Class Dpump Activate Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Dpump Deactivate

ux_device_class_dpump_deactivate

Icon 

Description

This event represents a USBX Device Class Dpump Deactivate Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Dpump Read

ux_device_class_dpump_read

Icon 

Description

This event represents a USBX Device Class Dpump Read Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Buffer.
- Info Field 3: Requested length.
- Info Field 4: Not used.

Device Class Dpump Write

ux_device_class_dpump_write

Icon 

Description

This event represents a USBX Device Class Dpump Write Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Data pointer.
- Info Field 3: Requested length.
- Info Field 4: Not used.

Device Class Hid Activate

ux_device_class_hid_activate

Icon 

Description

This event represents a USBX Device Class Hid Activate Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Hid Deactivate

ux_device_class_hid_deactivate

Icon 

Description

This event represents a USBX Device Class Hid Deactivate Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Hid Descriptor Send

ux_device_class_hid_descritpor_send

Icon 

Description

This event represents a USBX Device Class Hid Descriptor Send Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Descriptor type.
- Info Field 3: Request index.
- Info Field 4: Not used.

Device Class Hid Event Get

ux_device_class_hid_event_get

Icon 

Description

This event represents a USBX Device Class Hid Event Get Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Hid Event Set

ux_device_class_hid_event_set

Icon 

Description

This event represents a USBX Device Class Hid Event Set.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Hid Report Get

ux_device_class_hid_report_get

Icon 

Description

This event represents a USBX Device Class Hid Report Get Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Descriptor type.
- Info Field 3: Request index.
- Info Field 4: Not used.

Device Class Hid Report Set

ux_device_class_hid_report_set

Icon 

Description

This event represents a USBX Device Class Hid Report Set Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Descriptor type.
- Info Field 3: Request index.
- Info Field 4: Not used.

Device Class Pima Activate

ux_device_class_pima_activate

Icon 

Description

This event represents a USBX Device Class Pima Activate Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Pima Deactivate

ux_device_class_pima_deactivate

Icon 

Description

This event represents a USBX Device Class Pima Deactivate Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Pima Device Info Send

ux_device_class_pima_device_info_send

Icon 

Description

This event represents a USBX Device Class Pima Device Info Send Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.

Device Class Pima Event Get

ux_device_class_pima_event_get

Icon 

Description

This event represents a USBX Device Class Pima Event Get Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Pima event.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Pima Event Set

ux_device_class_pima_event_set

Icon 

Description

This event represents a USBX Device Class Pima Event Set Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Pima event.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Pima Object Add

ux_device_class_pima_object_add

Icon 

Description

This event represents a USBX Device Class Pima Object Add Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Object handle.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Pima Object Data Get

ux_device_class_pima_object_data_get

Icon 

Description

This event represents a USBX Device Class Pima Object Data Get Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Object handle.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Pima Object Data Send

ux_device_class_pima_object_data_send

Icon 

Description

This event represents a USBX Device Class Pima Object Data Send Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Object handle.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Pima Object Delete

ux_device_class_pima_object_delete

Icon 

Description

This event represents a USBX Device 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.

Device Class Pima Object Handles Send

ux_device_class_pima_object_handles_send

Icon 

Description

This event represents a USBX Device Class Pima Object Handles Send Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Storage ID.
- Info Field 3: Object format code.
- Info Field 4: Object association.

Device Class Pima Object Info Get

ux_device_class_pima_object_info_send

Icon 

Description

This event represents a USBX Device Class Pima Object Info Get Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Object handle.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Pima Object Info Send

ux_device_class_pima_object_info_send

Icon 

Description

This event represents a USBX Device Class Pima Object Info Send Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Pima Objects Number Send

ux_device_class_pima_object_number_send

Icon 

Description

This event represents a USBX Device Class Pima Object Number Send event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Storage ID.
- Info Field 3: Object format code.
- Info Field 4: Object associate.

Device Class Pima Partial Object Data Get

ux_device_class_pima_partial_object_data_get

Icon 

Description

This event represents a USBX Device Class Pima Partial Object Data Get Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Object handle.
- Info Field 3: Offset requested.
- Info Field 4: Length requested.

Device Class Pima Response Send

ux_device_class_pima_response_send

Icon 

Description

This event represents a USBX Device Class Pima Response Send Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Response code.
- Info Field 3: Number parameter.
- Info Field 4: Pima parameter 1.

Device Class Pima Storage Id Send

ux_device_class_pima_storage_id_send

Icon 

Description

This event represents a USBX Device Class Pima Storage Id Send Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Pima Storage Info Send

ux_device_class_pima_storage_info_send

Icon 

Description

This event represents a USBX Device Class Pima Storage Info Send Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Rndis Activate

ux_device_class_rndis_activate

Icon 

Description

This event represents a USBX Device Class Rndis Activate Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Rndis Deactivate

ux_device_class_rndis_deactivate

Icon 

Description

This event represents a USBX Device Class Rndis Deactivate Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Rndis Message Keep Alive

ux_device_class_rndis_msg_keep_alive

Icon 

Description

This event represents a USBX Device Class Rndis Message Keep Alive Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.

- Info Field 4: Not used.

Device Class Rndis Message Query

ux_device_class_rndis_msg_keep_query

Icon

Description

This event represents a USBX Device Class Rndis Message Query Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Rndis OID.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Rndis Message Reset

ux_device_class_rndis_msg_reset

Icon 

Description

This event represents a USBX Device Class Rndis Message Reset Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.

Device Class Rndis Message Set

ux_device_class_rndis_msg_set

Icon 

Description

This event represents a USBX Device Class Rndis Message Set Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Rndis OID.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Rndis Packet Receive

ux_device_class_rndis_packet_receive

Icon 

Description

This event represents a USBX Device Class Rndis Packet Receive Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.

- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Rndis Packet Transmit

ux_device_class_rndis_packet_transmit

Icon 

Description

This event represents a USBX Device Class Rndis Packet Transmit Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Storage Activate

ux_device_class_storage_activate

Icon 

Description

This event represents a USBX Device 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.

Device Class Storage Deactivate

ux_device_class_storage_deactivate

Icon 

Description

This event represents a USBX Device Class Storage Deactivate Event.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Storage Format

ux_device_class_storage_format

Icon 

Description

This event represents a USBX Device Class Storage Format 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 Inquiry

ux_device_class_storage_inquiry

Icon

Description

This event represents a USBX Device Class Storage Inquiry 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 Mode Select

ux_device_class_storage_mode_select

Icon 

Description

This event represents a USBX Device Class Storage Mode Select 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 Mode Sense

ux_device_class_storage_mode_sense

Icon 

Description

This event represents a USBX Device Class Storage Mode Sense Event.

Information Fields

- nfo Field 1: Class Instance.
- Info Field 2: Lun.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Class Storage Prevent Allow Media Removal

ux_device_class_storage_prevent_allow_media_removal

Icon 

Description

This event represents a USBX Device Class Storage Prevent Allow Media Removal 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 Read

ux_device_class_storage_read

Icon 

Description

This event represents a USBX Device Class Storage Read Event.

Information Fields

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

Device Class Storage Read Capacity

ux_device_class_storage_read_capacity

Icon 

Description

This event represents a USBX Device Class Storage Read Capacity 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 Read Format Capacity

ux_device_class_storage_read_format_capacity

Icon 

Description

This event represents a USBX Device Class Storage Read Format Capacity 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 Read TOC

ux_device_class_storage_read_toc

Icon 

Description

This event represents a USBX Device Class Storage Read TOC 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 Request Sense

ux_device_class_storage_request_sense

Icon 

Description

This event represents a USBX Device Class Storage Request Sense Event.

Information Fields

- 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

Icon 

Description

This event represents a USBX Device Class Storage Start Stop 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 Test Ready

ux_device_class_storage_test_ready

Icon 

Description

This event represents a USBX Device Class Storage Test Ready 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 Verify

ux_device_class_storage_verify

Icon 

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

ux_device_class_storage_write

Icon 

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

ux_device_class_alternate_setting_get

Icon 

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

ux_device_class_alternate_setting_set

Icon 

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

ux_device_stack_class_register

Icon 

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

ux_device_stack_clear_feature

Icon 

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.

Device Stack Configuration Get

ux_device_stack_configuration_get t

Icon 

Description

This event represents a USBX Device Stack Configuration Get Event.

Information Fields

- Info Field 1: Configuration value.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Stack Configuration Set

ux_device_stack_configuration_set

Icon 

Description

This event represents a USBX Device Stack Configuration Set Event.

Information Fields

- Info Field 1: Configuration value.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Stack Connect

ux_device_stack_connect

Icon 

Description

This event represents a USBX Device Stack Descriptor Send 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 Descriptor Send

ux_device_stack_descriptor_send

Icon 

Description

This event represents a USBX Device Stack Descriptor Send Event.

Information Fields

- Info Field 1: Descriptor type.
- Info Field 2: Request index.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Stack Disconnect

ux_device_stack_disconnect

Icon 

Description

This event represents a USBX Device Stack Disconnect Event.

Information Fields

- Info Field 1: Device.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Device Stack Endpoint Stall

ux_device_stack_endpoint_stall

Icon 

Description

This event represents a USBX Device Stack Endpoint Stall Event.

Information Fields

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

Device Stack Get Status

ux_device_stack_get_status

Icon 

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

ux_device_stack_host_wakeup

Icon 

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

ux_device_stack_initialize

Icon 

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

ux_device_stack_interface_delete

Icon 

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

ux_device_stack_interface_get

Icon 

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

ux_device_stack_interface_set

Icon 

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.

Device Stack Set Feature

ux_device_stack_set_feature

Icon 

Description

This event represents a USBX Device Stack Set Feature Event.

Information Fields

- 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

Icon 

Description

This event represents a USBX Device Stack Transfer Abort Event.

Information Fields

- 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

Icon 

Description

This event represents a USBX Device Stack Transfer All Request Abort Event.

Information Fields

- 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

Icon 

Description

This event represents a USBX Device Stack Transfer Request Event.

Information Fields

- 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

Icon 

Description

This event represents a USBX Host Class Asix Activate.

Information Fields

- 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

Icon 

Description

This event represents a USBX Host Class Asix 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 Asix Interrupt Notification

ux_host_class_asix_interrupt_notification

Icon 

Description

This event represents a USBX Host Class Asix Interrupt Notification 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 Asix Read

ux_host_class_asix_read

Icon 

Description

This event represents a USBX Host Class Asix 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 Asix Write

ux_host_class_asix_write

Icon 

Description

This event represents a USBX Host Class Asix 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 Audio Activate

ux_host_class_audio_activate

Icon 

Description

This event represents a USBX Host Class Audio 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 Audio Control Value Get

ux_host_class_audio_control_value_get

Icon 

Description

This event represents a USBX Host Class Audio Control Value 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 Audio Control Value Set

ux_host_class_audio_control_value_set

Icon 

Description

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

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Audio control.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Audio Deactivate

ux_host_class_audio_deactivate

Icon 

Description

This event represents a USBX Host Class Audio 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 Audio Read

ux_host_class_audio_read

Icon 

Description

This event represents a USBX Host Class Audio 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 Audio Streaming Sampling Get

ux_host_class_audio_streaming_sampling_get

Icon 

Description

This event represents a USBX Host Class Audio Streaming Sampling 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 Audio Streaming Sampling Set

ux_host_class_audio_streaming_sampling_set

Icon 

Description

This event represents a USBX Host Class Audio Streaming Sampling Set Event.

Information Fields

- 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

Icon 

Description

This event represents a USBX Host Class Audio 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 Cdc Acm Activate

ux_host_class_cdc_acm_activate

Icon 

Description

This event represents a USBX Host Class Cdc Acm 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 Cdc Acm Deactivate

ux_host_class_cdc_acm_deactivate

Icon 

Description

This event represents a USBX Host Class Cdc Acm 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 Cdc Acm Ioctl Abort In Pipe

ux_host_class_cdc_acm_ioctl_abort_in_pipe

Icon 

Description

This event represents a USBX Host Class Cdc Acm Ioctl Abort In Pipe Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Endpoint.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Cdc Acm Ioctl Abort Out Pipe

ux_host_class_cdc_acm_ioct_abort_out_pipe

Icon ![Host Class C D C A C M I O C T L Abort Out Pipe icon

Description

This event represents a USBX Host Class Cdc Acm Ioctl Abort Out Pipe Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Endpoint.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Cdc Acm Ioctl Get Device Status

ux_host_class_cdc_acm_ioctl_get_device_status

Icon 

Description

This event represents a USBX Host Class Cdc Acm Ioctl Get Device Status Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Device status.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Cdc Acm Ioctl Get Line Coding

ux_host_class_cdc_acm_ioctl_get_line_coding

Icon 

Description

This event represents a USBX Host Class Cdc Acm Ioctl Get Line Coding Event.

Information Fields

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

Host Class Cdc Acm Ioctl Notification Callback

ux_host_class_cdc_acm_ioctl_notification_callback

Icon 

Description

This event represents a USBX Host Class Cdc Acm Ioctl Notification Callback Event.

Information Fields

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

Host Class Cdc Acm Ioctl Send Break

ux_host_class_cdc_acm_ioctl_send_break

Icon 

Description

This event represents a USBX Host Class Cdc Acm Ioctl Send Break Event.

Information Fields

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

Host Class Cdc Acm Ioctl Set Line Coding

ux_host_class_cdc_acm_ioctl_set_line_coding

Icon  icon](./media/user-guide/usbx-events/image98.png)

Description

This event represents a USBX Host Class Cdc Acm Ioctl Set Line Coding Event.

Information Fields

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

Host Class Cdc Acm Ioctl Set Line State

ux_host_class_cdc_acm_ioctl_set_line_state

Icon 

Description

This event represents a USBX Host Class Cdc Acm 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 Cdc Acm Read

ux_host_class_cdc_acm_read

Icon 

Description

This event represents a USBX Host Class Cdc Acm 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 Cdc Acm Reception Start

ux_host_class_cdc_acm_reception_start

Icon 

Description

This event represents a USBX Host Class Cdc Acm 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 Cdc Acm Reception Stop

ux_host_class_cdc_acm_reception_stop

Icon 

Description

This event represents a USBX Host Class Cdc Acm Reception Stop Event.

Information Fields

- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Cdc Acm Write

ux_host_class_acm_write

Icon 

Description

This event represents a USBX Host Class Cdc Acm 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 Dpump Activate

ux_host_class_dpump_activate

Icon 

Description

This event represents a USBX Host Class Dpump 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 Dpump Deactivate

ux_host_class_dpump_deactivate

Icon 

Description

This event represents a USBX Host Class Dpump 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 Dpump Read

ux_host_class_dpump_read

Icon 

Description

This event represents a USBX Host Class Dpump 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 Dpump Write

ux_host_class_dpump_write

Icon 

Description

This event represents a USBX Host Class Dpump 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 Hid Activate

ux_host_class_hid_activate

Icon 

Description

This event represents a USBX Host Class Hid 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 Hid Client Register

ux_host_class_hid_client_register

Icon 

Description

This event represents a USBX Host Class Hid Client Register Event.

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

Icon 

Description

This event represents a USBX Host Class Hid 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 Hid Idle Get

ux_host_class_hid_idle_get

Icon 

Description

This event represents a USBX Host Class Hid Idle 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 Hid Idle Set

ux_host_class_hid_idle_set

Icon 

Description

This event represents a USBX Host Class Hid Idle Set 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 Hid Keyboard Activate

ux_host_class_hid_keyboard_activate

Icon 

Description

This event represents a USBX Host Class Hid Keyboard Activate Event.

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

Icon 

Description

This event represents a USBX Host Class Hid Keyboard Deactivate Event.

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 Mouse Activate

ux_host_class_hid_mouse_activate

Icon 

Description

This event represents a USBX Host Class Hid Mouse Activate Event.

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 Mouse Deactivate

ux_host_class_hid_mouse_deactivate

Icon 

Description

- This event represents a USBX Host Class Hid Mouse Deactivate Event.

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 Remote Control Activate

ux_host_class_hid_remote_control_activate

Icon 

Description

This event represents a USBX Host Class Hid Remote Control Activate Event.

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 Remote Control Deactivate

ux_host_class_hid_remote_control_deactivate

Icon 

Description

This event represents a USBX Host Class Hid Remote Control Deactivate Event.

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 Report Get

ux_host_class_hid_report_get

Icon 

Description

This event represents a USBX Host Class Hid Report Get.

Information Fields

- 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

Icon 

Description This event represents a USBX Host Class Hid Report Set.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Client report.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Hub Activate

ux_host_class_hub_activate

Icon 

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

ux_host_class_hub_change_detect

Icon 

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

ux_host_class_hub_deactivate

Icon 

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

ux_host_class_hub_change_connection_process

Icon 

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

ux_host_class_hub_port_change_enable_process

Icon 

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

ux_host_class_hub_port_change_over_current_process

Icon 

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.

Host Class Hub Port Change Reset Process

ux_host_class_hub_port_change_reset_process

Icon 

Description

This event represents a USBX Host Class Hub Port Change Reset Process Event.

Information Fields

- Info Field 1: Hub. Info Field 2: Port.
- Info Field 3: Port status.
- Info Field 4: Not used.

Host Class Hub Port Change Suspend Process

ux_host_class_hub_port_change_suspend_process

Icon 

Description

This event represents a USBX Host Class Hub Port Change Suspend 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 Pima Activate

ux_host_class_pima_activate

Icon 

Description

This event represents a USBX Host Class Pima 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 Pima Deactivate

ux_host_class_pima_deactivate

Icon 

Description

This event represents a USBX Host Class Pima 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 Pima Device Info Get

ux_host_class_pima_device_info_get

Icon 

Description

This event represents a USBX Host Class Pima Device Info Get Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Pima device.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Pima Device Reset

ux_host_class_pima_device_reset

Icon 

Description

This event represents a USBX Host Class Pima Device Reset 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 Notification

ux_host_class_pima_notification

Icon 

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

ux_host_class_pima_number_objects_get

Icon 

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

ux_host_class_pima_object_close

Icon 

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

ux_host_class_pima_object_copy

Icon 

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

ux_host_class_pima_object_delete

Icon 

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

ux_host_class_pima_object_get

Icon 

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.

Host Class Pima Object Info Get

ux_host_class_pima_object_info_get

Icon 

Description

This event represents a USBX Host Class Pima Object Info Get Event.

Information Fields

- 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 

Description

This event represents a USBX Host Class Pima Object Info Send 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 Move

ux_host_class_pima_object_move

Icon 

Description

This event reprThis event represents a USBX Host Class Pima Object Move Event.

Information Fields

- 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_move

Icon 

Description

This event represents a USBX Host Class Pima Object Send Event.

Information Fields

- 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 

Description

This event represents a USBX Host Class Pima Object Transfer Abort Event.

Information Fields

- 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 

Description

This event represents a USBX Host Class Pima Read Event.

Information Fields

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

Host Class Pima Request Cancel

ux_host_class_pima_request_cancel

Icon 

Description

This event represents a USBX Host Class Pima Request Cancel 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 Session Close

ux_host_class_pima_session_close

Icon 

Description

This event represents a USBX Host Class Pima Session Close Event.

Information Fields

- 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

ux_host_class_pima_session_open

Icon 

Description This event represents a USBX Host Class Pima Session Open Event.

Information Fields

- 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

ux_host_class_pima_session_ids_get

Icon 

Description

This event represents a USBX Host Class Pima Storage Ids Get Event.

Information Fields

- Info Field 1: Class instance.
- nfo Field 2: Storage ID array.
- Info Field 3: Storage ID length. Info Field 4: Not used.

Host Class Pima Storage Info Get

ux_host_class_pima_storage_info_get

Icon 

Description

This event represents a USBX Host Class Pima Storage Info Get Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Storage ID.
- Info Field 3: Storage.
- Info Field 4: Not used.

Host Class Pima Thumb Get

ux_host_class_pima_thumb_get

Icon 

Description

This event represents unaccepting a TCP server connection via nx_tcp_server_socket_unaccept.

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 Write

ux_host_class_pima_thumb_get

Icon 

Description

This event represents a USBX Host Class Pima Write.

Information Fields

- Info Field 1: Class Instance.
- Info Field 2: Data pointer.
- Info Field 3: Data length.
- Info Field 4: Not used.

Host Class Printer Activate

ux_host_class_printer_activate

Icon 

Description

This event represents a USBX Host Class Printer Activate Event.

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.

Host Class Printer Deactivate

ux_host_class_printer_deactivate

Icon 

Description

This event represents a USBX Host Class Printer 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 Printer Name Get

ux_host_class_printer_name_get

Icon 

Description

This event represents a USBX Host Class Printer Name 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 Printer Read

ux_host_class_printer_read

Icon 

Description

This event represents a USBX Host Class Printer 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 Printer Soft Reset

ux_host_class_printer_soft_reset

Icon 

Description

This event represents a USBX Host Class Printer Soft Reset 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 Printer Status Get

ux_host_class_printer_status_get

Icon 

Description

This event represents a USBX Host Class Printer Status Get Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Printer status.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Printer Write

ux_host_class_printer_write

Icon 

Description

This event represents a USBX Host Class Printer Write.

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 Activate

ux_host_class_prolific_activate

Icon 

Description

This event represents a USBX Host Class Prolific 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 Prolific Deactivate

ux_host_class_prolific_deactivate

Icon 

Description

This event represents a USBX Host Class Prolific 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 Prolific Ioctl Abort In Pipe

ux_host_class_prolific_ioctl_abort_in_pipe

Icon 

Description

This event represents a USBX Host Class Prolific Ioctl Abort In Pipe Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Endpoint.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Prolific Ioctl Abort Out Pipe

ux_host_class_prolific_ioctl_abort_out_pipe

Icon 

Description

This event represents a USBX Host Class Prolific Ioctl Abort Out Pipe Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Endpoint.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Prolific Ioctl Get Device Status

ux_host_class_prolific_ioctl_get_device_status

Icon 

Description

This event represents a USBX Host Class Prolific Ioctl Get Device Status Event.

Information Fields

- 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

ux_host_class_prolific_ioctl_get_line_coding

Icon 

Description

This event represents a USBX Host Class Prolific Ioctl Get Line Coding 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 Ioctl Purge

ux_host_class_prolific_ioctl_purge

Icon 

Description

This event represents a USBX Host Class Prolific Ioctl Purge 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 Ioctl Report Device

ux_host_class_prolific_ioctl_report_device

Icon 

Description

This event represents a USBX Host Class Prolific Ioctl Report Device Status Change 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 Ioctl Send Break

ux_host_class_prolific_ioctl_send_break

Icon 

Description

This event represents a USBX Host Class Prolific Ioctl Send Break 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 Ioctl Set Line Coding

ux_host_class_prolific_ioctl_set_line_coding

Icon 

Description

This event represents a USBX Host Class Prolific Ioctl Set Line Coding 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 Ioctl Set Line State

ux_host_class_prolific_ioctl_set_line_state

Icon 

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

ux_host_class_prolific_read

Icon 

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

ux_host_class_prolific_reception_start

Icon 

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

ux_host_class_prolific_reception_stop

Icon 

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

ux_host_class_prolific_write

Icon 

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

ux_host_class_storage_activate

Icon 

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.

Host Class Storage Deactivate

ux_host_class_storage_deactivate

Icon 

Description

This event represents a USBX Host Class Storage 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 Storage Media Capacity Get

ux_host_class_storage_media_capacity_get

Icon 

Description

This event represents a USBX Host Class Storage Media Capacity 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 Storage Media Format Capacity Get

ux_host_class_storage_media_format_capacity_get

Icon 

Description

This event represents a USBX Host Class Storage Media Format Capacity 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 Storage Media Mount

ux_host_class_storage_media_mount

Icon 

Description

This event represents a USBX Host Class Storage Media Mount Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Sector.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Storage Media Open

ux_host_class_storage_media_open

Icon 

Description

This event represents a USBX Host Class Storage Media Open Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Media.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Class Storage Media Read

ux_host_class_storage_media_read

Icon 

Description

This event represents a USBX Host Class Storage Media Read Event.

Information Fields

- Info Field 1: Class instance.
- Info Field 2: Sector start.
- Info Field 3: Sector count.
- Info Field 4: Data pointer.

Host Class Storage Media Write

ux_host_class_storage_media_write

Icon 

Description

This event represents a USBX Host Class Storage Media Write Event.

Information Fields

- 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

ux_host_class_storage_request_sense

Icon 

Description

This event represents a USBX Host Class Storage Request Sense 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 Storage Start Stop

ux_host_class_storage_start_stop

Icon 

Description

This event represents a USBX Host Class Storage Start Stop Event.

Information Fields

- 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

ux_host_class_storage_unit_ready_test

Icon 

Description

This event represents a USBX Host Class Storage Unit Ready Test Event.

Information Fields

- 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

ux_host_class_instance_create

Icon 

Description

This event represents a USBX Host Stack Class Instance Create Event.

Information Fields

- Info Field 1: Class.
- Info Field 2: Class Instance.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Stack Class Instance Destroy

ux_host_class_instance_create

Icon 

Description

This event represents a USBX Host Stack Class Instance Destroy Event.

Information Fields

- Info Field 1: Class.
- Info Field 2: Class Instance.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Stack Configuration Delete

ux_host_class_configuration_delete

Icon 

Description

This event represents a USBX Host Stack Configuration Delete Event.

Information Fields

- Info Field 1: Configuration.
- Info Field 2: Not used.
- Info Field 3: Not used.

- Info Field 4: Not used.

Host Stack Configuration Enumerate

ux_host_stack_configuration_enumerate

Icon 

Description

This event represents a USBX Host Stack Configuration Enumerate Event.

Information Fields

- Info Field 1: Device.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Stack Configuration Instance Create

ux_host_stack_configuration_instance_create

Icon 

Description

This event represents a USBX Host Stack Configuration Instance Create Event.

Information Fields

- Info Field 1: Configuration.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Stack Configuration Instance Delete

ux_host_stack_configuration_instance_delete

Icon 

Description

This event represents a USBX Host Stack Configuration Instance Delete Event.

Information Fields

- Info Field 1: Configuration.
- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Stack Configuration Set

ux_host_stack_configuration_set

Icon 

Description

This event represents a USBX Host Stack Configuration Set Event.

Information Fields

- Info Field 1: Configuration.

- Info Field 2: Not used.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Stack Device Address Set

ux_host_stack_device_address_set

Icon 

Description

This event represents a USBX Host Stack Device Address Set Event.

Information Fields

- Info Field 1: Device.
- Info Field 2: Device Address.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Stack Device Configuration Get

ux_host_stack_device_configuration_get

Icon 

Description

This event represents a USBX Host Stack Device Configuration Get Event.

Information Fields

- Info Field 1: Device.
- nfo 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 

Description

This event represents a USBX Host Stack Device Configuration Select Event.

Information Fields

- 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 

Description

This event represents a USBX Host Stack Device Descriptor Read Event.

Information Fields

- 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 

Description

This event represents a USBX Host Stack Device Get Event.

Information Fields

- 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 

Description

This event represents a USBX Host Stack Device Remove Event.

Information Fields

- 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 

Description

This event represents a USBX Host Stack Device Resource Free Event.

Information Fields

- 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 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.

Host Stack Interface Endpoint Get

Interface_ TCP retry entry

Icon 

Description

This event represents an internal NetX TCP retry event.

Information Fields

- Info Field 1: Interface.
- Info Field 2: Endpoint index.
- Info Field 3: Not used.
- Info Field 4: Not used.

Host Stack Interface Instance Create

ux_host_stack_interface_instance_create

Icon 

Description

This event represents a USBX Host Stack Interface Instance Create Event.

Information Fields

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

Host Stack Interface Instance Delete

ux_host_stack_interface_instance_delete

Icon 

Description

This event represents a USBX Host Stack Interface Instance Delete Event.

Information Fields

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

Host Stack Interface Set

ux_host_stack_interface_set

Icon 

Description

This event represents a USBX Host Stack Interface Set Event.

Information Fields

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

Host Stack Interface Setting Select

ux_host_stack_interface_setting_select

Icon 

Description

This event represents a USBX Host Stack Interface Setting Select Event.

Information Fields

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

Host Stack New Configuration Create

ux_host_stack_new_configuration_create

Icon 

Description

This event represents a USBX Host Stack New Configuration Create Event.

Information Fields

- Info Field 1: Device.
- Info Field 2: Configuration.
- Info Field 3: Not used.
- Info Field 4: Not used.

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

Internal I/O driver get status

Icon 

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.

USBX Error

ux_error

Icon 

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.

Chapter 10 - Customer user events

7/20/2020 • 3 minutes to read

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

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:

```
tx_trace_user_event_insert(4096, 1, 2, 3, 4);
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 452 and 453, which were placed in the event buffer via the previous ***tx_trace_user_event_insert*** examples.

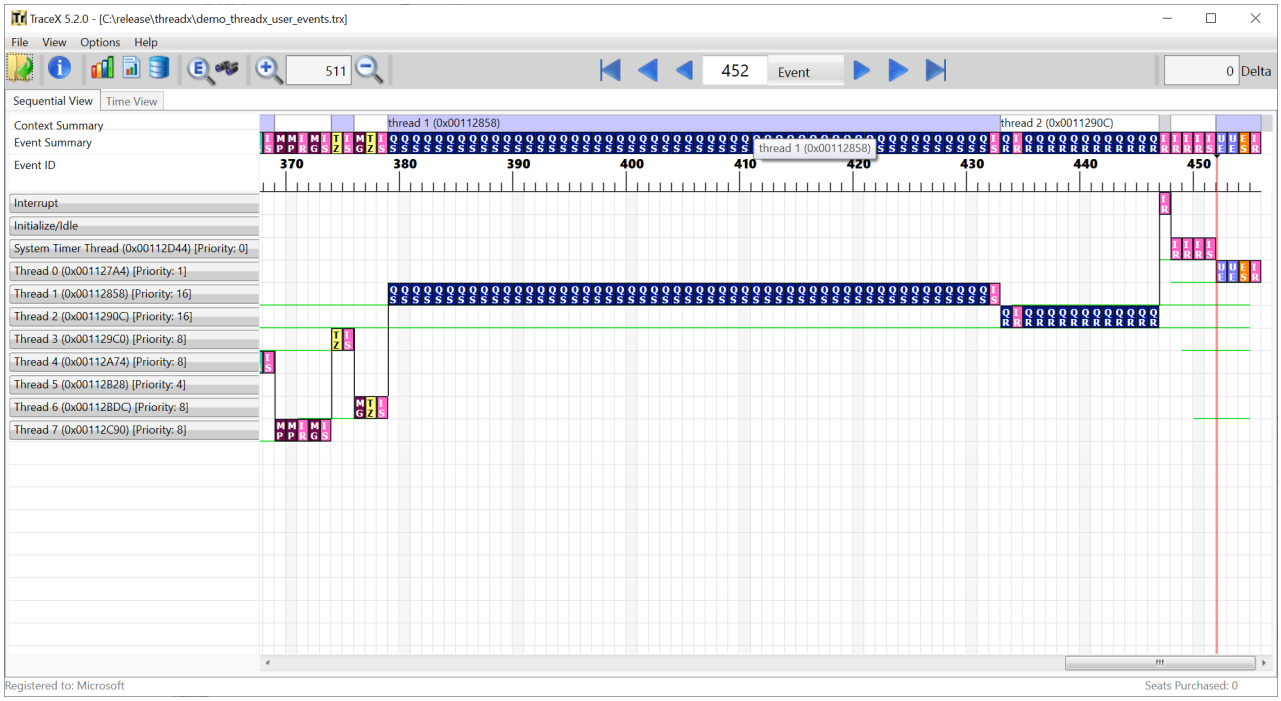


FIGURE 10.1

Detailed information is also available for user-defined Events. Figure 10.2 shows the detailed event information for event 452, which has event number 4096 and shows the specified four information fields.

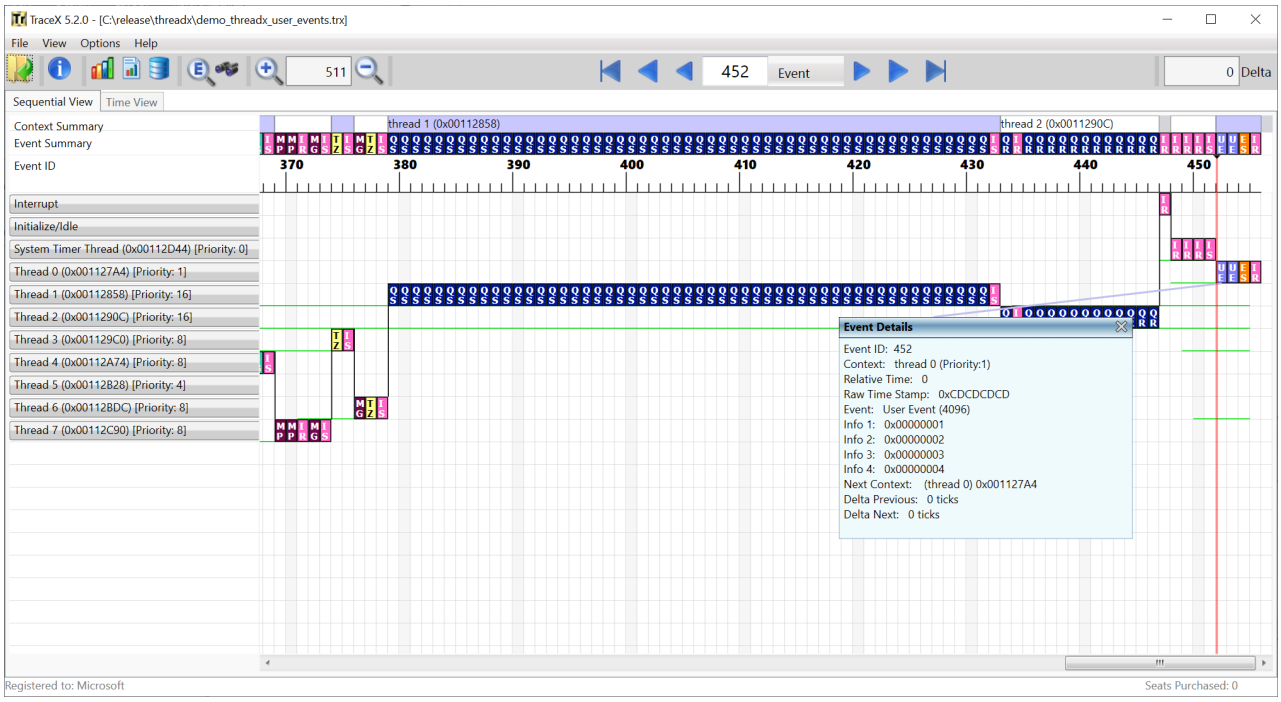


FIGURE 10.2

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*** subdirectory in the main TraceX installation directory. An example directory path is shown in Figure 10.3.

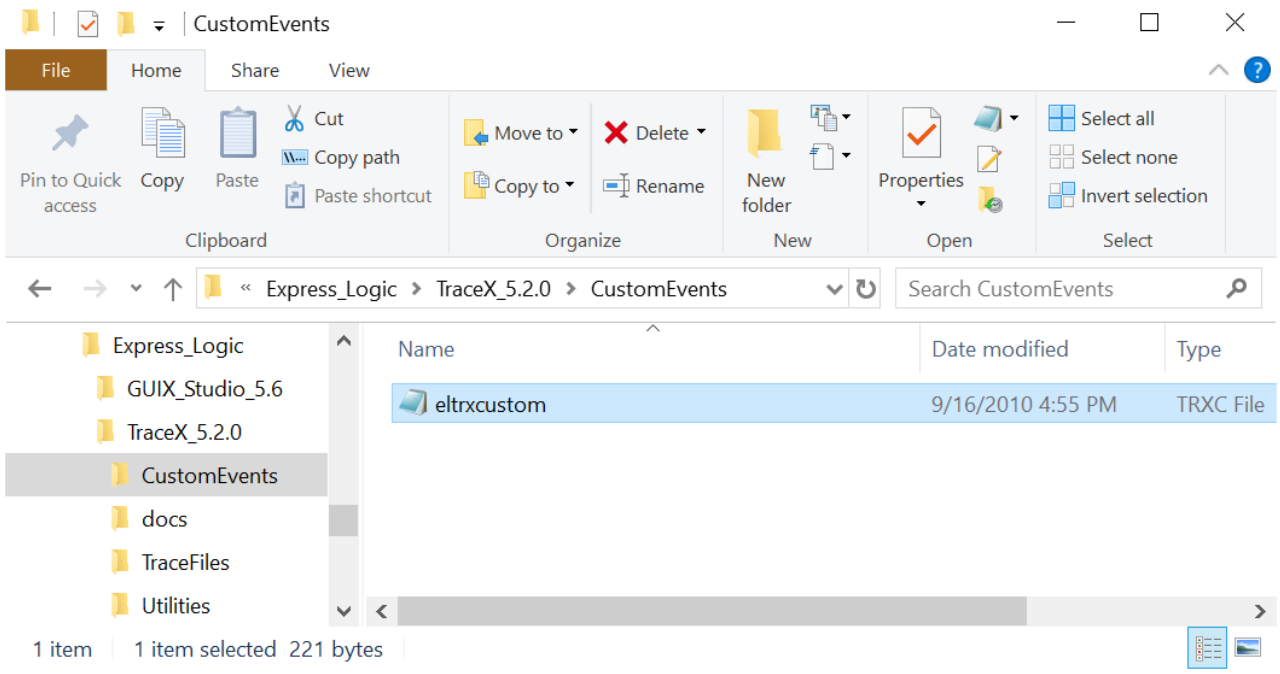


FIGURE 10.3

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:

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

Each line between Start and End is used to define 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:

number, name, abbreviation, top_color, bottom_color, label1, label2, label2, label4

where:

- number: Defines the user-defined event number, between 4096 and 65535, inclusive.
- name: Defines the logical name for the userdefined 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
  - BLACK = (0,0,0)
  - WHITE = (255,255,255)
  - RED = (255,0,0)
  - GREEN = (0,255,0)
  - BLUE = (0,0,255)
  - YELLOW = (255,255,0)
  - CYAN = (0,255,255)
  - 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:
    https://en.wikipedia.org/wiki/RGB#Digital_representations
- botton_color: Defines the RGB value for the bottomhalf of the icon, which is a three digit number in parenthesis.
- label1: Label for *info_field_1*, as specified in the *tx_trace_user_event_insert* call.
- label2: Label for *info_field_2*, as specified in the *tx_trace_user_event_insert* call.
- label3: Label for *info_field_3*, as specified in the *tx_trace_user_event_insert* call.
- label4: Label for *info_field_4*, as specified in the *tx_trace_user_event_insert* call.

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 userdefined 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***.

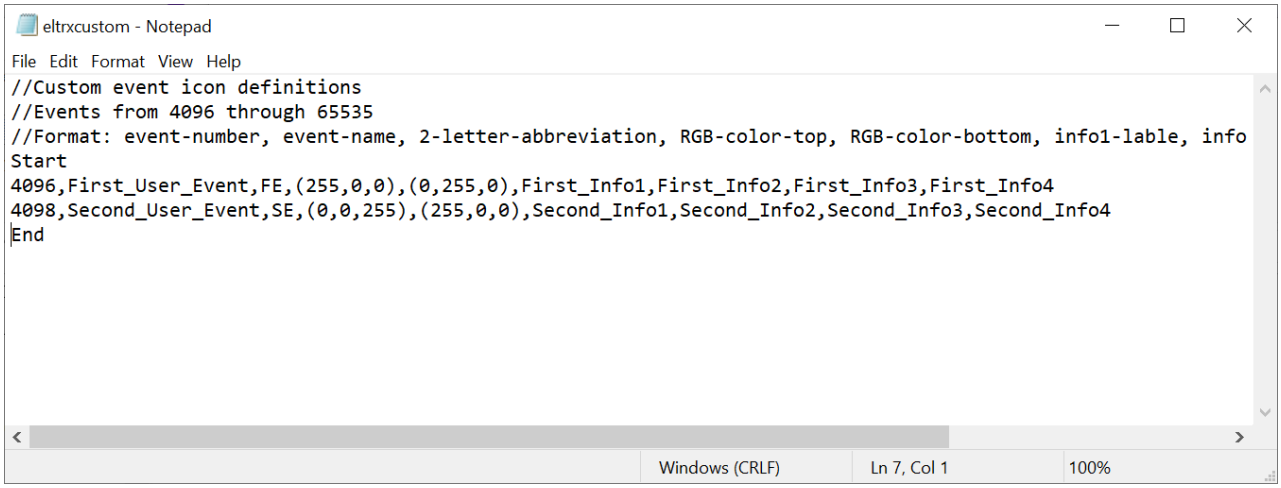


FIGURE 10.4

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 userdefined events 452 and 453

with the custom event icons defined in ***eltrxcustom.trxc***.

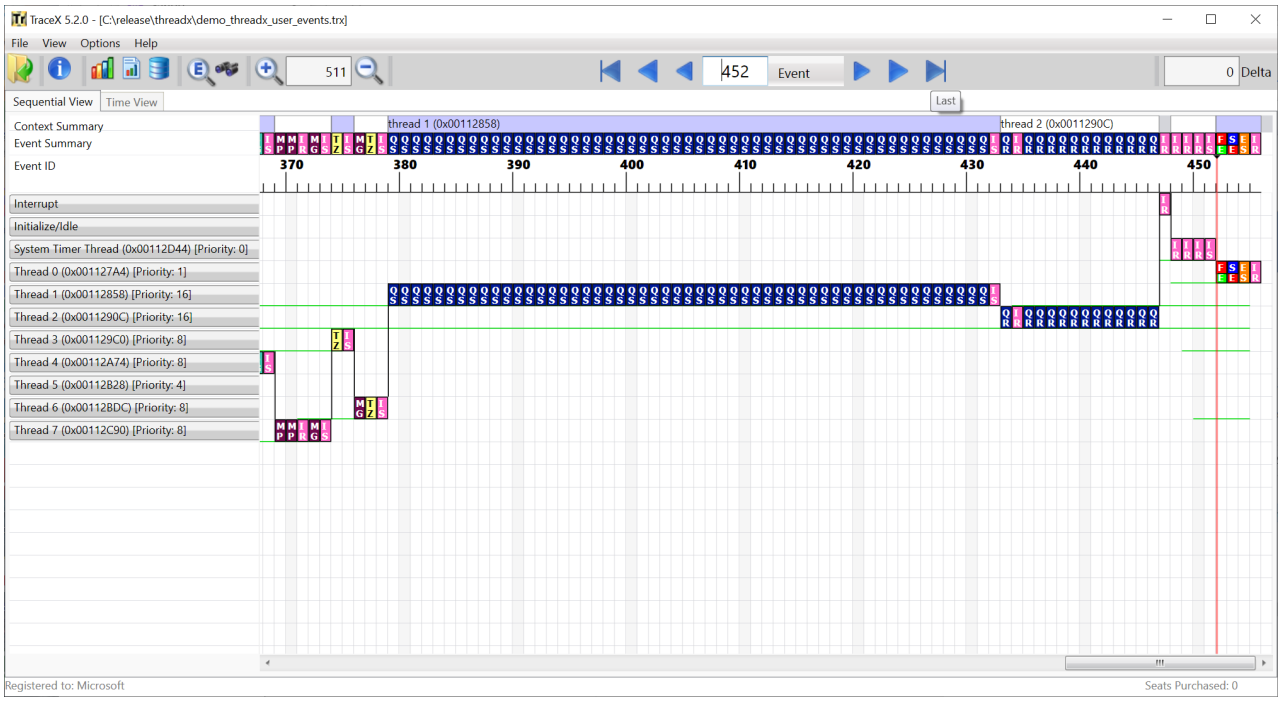


FIGURE 10.5

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 452. Note that the event name and information fields all match the sample definition that was added to ***eltrxcustom.trxc***.

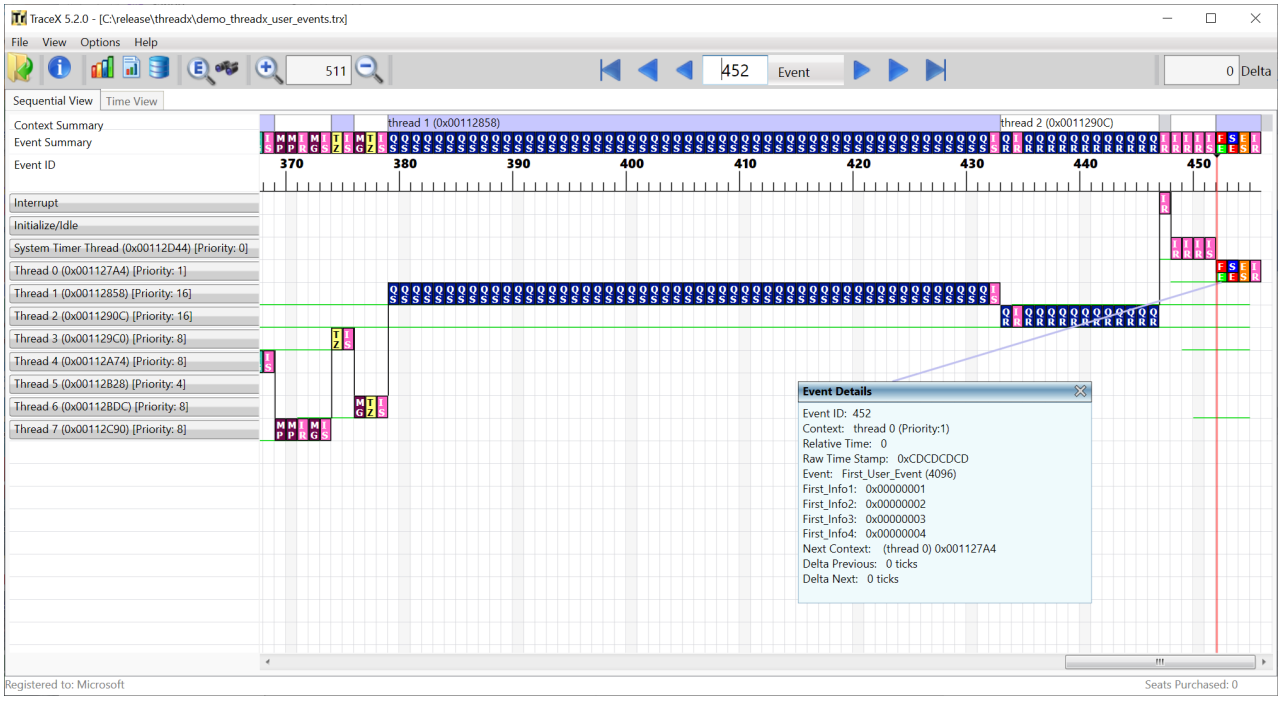


FIGURE 10.6

Chapter 11 - Format of event trace buffer

7/20/2020 • 6 minutes to read

Azure RTOS 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

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 trace buffer resides. The following data structure defines the 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_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 byes 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 timestamp 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***.

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 bufferrelative 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 im../mediately 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 im../mediately 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:

```
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_thread_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:

VALUE	OBJECT TYPE
0	Not Valid
1	Thread
2	Timer
3	Queue
4	Semaphore
5	Mutex
6	Event Flags Group
7	Block Pool
8	Byte Pool
9	../media
10	File
11	IP
12	Packet Pool
13	TCP Socket
14	UDP Socket
15-20	Reserved
21	USB Host Stack Device
22	USB Host Stack Interface
23	USB Host Endpoint
24	USB Host Class
25	USB Device
26	USB Device Interface
27	USB Device Endpoint

VALUE	OBJECT TYPE
28	USB Device Class

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:

OBJECT TYPE	PARAMETER 1	PARAMETER 2
Thread	Stack Start	Stack Size
Timer	Initial Ticks	Reschedule Ticks
Queue	Queue Size	Message Size
Semaphore	Initial Instances	-
Mutex	Inheritance Flag	-
Event Flags Group	-	-
Block Pool	Total Blocks	Block Size
Byte Pool	Total Bytes	-
../media	Fat Cache Size	Sector Cache Size
File	-	-
IP	Stack Start	Stack Size
Packet Pool	Packet Size	Number of Packets
TCP Socket	IP address	Window Size
UDP Socket	IP address	RX Queue Max

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:

```
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.

Appendix A - Sample tx_port.h

5/21/2020 • 8 minutes to read

This chapter displays a sample **tx_port.h** file.

```
/**************************************************************************/
/*                                                                        */
/*       Copyright (c) Microsoft Corporation. All rights reserved.        */
/*                                                                        */
/*       This software is licensed under the Microsoft Software License   */
/*       Terms for Microsoft Azure RTOS. Full text of the license can be  */
/*       found in the LICENSE file at https://aka.ms/AzureRTOS_EULA       */
/*       and in the root directory of this software.                      */
/*                                                                        */
/**************************************************************************/


/**************************************************************************/
/**************************************************************************/
/**                                                                       */
/** ThreadX Component                                                     */
/**                                                                       */
/**   Port Specific                                                       */
/**                                                                       */
/**************************************************************************/
/**************************************************************************/


/**************************************************************************/
/*                                                                        */
/*  PORT SPECIFIC C INFORMATION                            RELEASE        */
/*                                                                        */
/*    tx_port.h                                          Cortex-M4/IAR     */
/*                                                            6.0         */
/*                                                                        */
/*  AUTHOR                                                                */
/*                                                                        */
/*    William E. Lamie, Microsoft Corporation                             */
/*                                                                        */
/*  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             */
/*                                                                        */
/*  05-19-2020     William E. Lamie         Initial Version 6.0           */
/*                                                                        */
/**************************************************************************/

#ifndef TX_PORT_H
#define TX_PORT_H
```

```
/* Determine if the optional ThreadX user define file should be used.  */

#ifdef 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.  */

#include "tx_user.h"
#endif


/* Define compiler library include files.  */

#include <stdlib.h>
#include <string.h>
#include <intrinsics.h>
#ifdef  TX_ENABLE_IAR_LIBRARY_SUPPORT
#include <yvals.h>
#endif


/* 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 Cortex-M3 port.  */

#define TX_INT_DISABLE                          1           /* Disable interrupts               */
#define TX_INT_ENABLE                           0           /* Enable 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                    *((ULONG *) 0x0a800024)
#define TX_TRACE_TIME_MASK                      0x0000FFFFUL

*/

#ifndef TX_MISRA_ENABLE
#ifndef TX_TRACE_TIME_SOURCE
#define TX_TRACE_TIME_SOURCE                    *((ULONG *) 0xE0001004)
#endif
#else
ULONG   _tx_misra_time_stamp_get(VOID);
#define TX_TRACE_TIME_SOURCE                    _tx_misra_time_stamp_get()
#endif

#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.  */

#define TX_PORT_SPECIFIC_BUILD_OPTIONS          (0)


/* Define the in-line initialization constant so that modules with in-line
   initialization capabilities can prevent their initialization from being
   a function call.  */

#ifdef TX_MISRA_ENABLE
#define TX_DISABLE_INLINE
#else
#define TX_INLINE_INITIALIZATION
#endif


/* 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.  */

#ifndef TX_MISRA_ENABLE
#ifdef TX_ENABLE_STACK_CHECKING
#undef TX_DISABLE_STACK_FILLING
#endif
#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
#ifdef  TX_ENABLE_IAR_LIBRARY_SUPPORT
#define TX_THREAD_EXTENSION_2           VOID    *tx_thread_iar_tls_pointer;
#else
#define TX_THREAD_EXTENSION_2
#endif
#ifndef TX_ENABLE_EXECUTION_CHANGE_NOTIFY
#define TX_THREAD_EXTENSION_3
#else
#define TX_THREAD_EXTENSION_3           unsigned long long  tx_thread_execution_time_total; \
                                        unsigned long long  tx_thread_execution_time_last_start;
#endif
```

```
/* 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.  */


#ifdef  TX_ENABLE_IAR_LIBRARY_SUPPORT
#if (__VER__ < 8000000)
#define TX_THREAD_CREATE_EXTENSION(thread_ptr)                      thread_ptr -> tx_thread_iar_tls_pointer =
__iar_dlib_perthread_allocate();
#define TX_THREAD_DELETE_EXTENSION(thread_ptr)                      __iar_dlib_perthread_deallocate(thread_ptr
-> tx_thread_iar_tls_pointer); \
                                                                    thread_ptr -> tx_thread_iar_tls_pointer =
TX_NULL;
#define TX_PORT_SPECIFIC_PRE_SCHEDULER_INITIALIZATION               __iar_dlib_perthread_access(0);
#else
void    *_tx_iar_create_per_thread_tls_area(void);
void    _tx_iar_destroy_per_thread_tls_area(void *tls_ptr);
void    __iar_Initlocks(void);

#define TX_THREAD_CREATE_EXTENSION(thread_ptr)                      thread_ptr -> tx_thread_iar_tls_pointer =
_tx_iar_create_per_thread_tls_area();
#define TX_THREAD_DELETE_EXTENSION(thread_ptr)                      do
{_tx_iar_destroy_per_thread_tls_area(thread_ptr -> tx_thread_iar_tls_pointer); \
                                                                        thread_ptr ->
tx_thread_iar_tls_pointer =  TX_NULL; } while(0);
#define TX_PORT_SPECIFIC_PRE_SCHEDULER_INITIALIZATION               do {__iar_Initlocks();} while(0);
#endif
#else
#define TX_THREAD_CREATE_EXTENSION(thread_ptr)
#define TX_THREAD_DELETE_EXTENSION(thread_ptr)
#endif


#ifdef __ARMVFP__

#ifdef TX_MISRA_ENABLE

ULONG  _tx_misra_control_get(void);
void   _tx_misra_control_set(ULONG value);
ULONG  _tx_misra_fpccr_get(void);
void   _tx_misra_vfp_touch(void);

#endif

/* A completed thread falls into _thread_shell_entry and we can simply deactivate the FPU via CONTROL.FPCA
   in order to ensure no lazy stacking will occur. */

#ifndef TX_MISRA_ENABLE
```

```
#define TX_THREAD_COMPLETED_EXTENSION(thread_ptr)                   {
\
                                                                    ULONG  _tx_vfp_state;
\
                                                                        _tx_vfp_state =  __get_CONTROL();
\
                                                                        _tx_vfp_state =  _tx_vfp_state & ~
((ULONG) 0x4);                                          \
                                                                        __set_CONTROL(_tx_vfp_state);
\
                                                                    }
#else

#define TX_THREAD_COMPLETED_EXTENSION(thread_ptr)                   {
\
                                                                    ULONG  _tx_vfp_state;
\
                                                                        _tx_vfp_state =
_tx_misra_control_get();                                             \
                                                                        _tx_vfp_state =  _tx_vfp_state & ~
((ULONG) 0x4);                                          \
                                                                        _tx_misra_control_set(_tx_vfp_state);
\
                                                                    }

#endif

/* A thread can be terminated by another thread, so we first check if it's self-terminating and not in an ISR.
   If so, deactivate the FPU via CONTROL.FPCA. Otherwise we are in an interrupt or another thread is
terminating
   this one, so if the FPCCR.LSPACT bit is set, we need to save the CONTROL.FPCA state, touch the FPU to flush
   the lazy FPU save, then restore the CONTROL.FPCA state. */

#ifndef TX_MISRA_ENABLE

#define TX_THREAD_TERMINATED_EXTENSION(thread_ptr)                  {
\
                                                                    ULONG  _tx_system_state;
\
                                                                        _tx_system_state =
TX_THREAD_GET_SYSTEM_STATE();                                         \
                                                                        if ((_tx_system_state == ((ULONG) 0))
&& ((thread_ptr) == _tx_thread_current_ptr))        \
                                                                        {
\
                                                                        ULONG  _tx_vfp_state;
\
                                                                            _tx_vfp_state =  __get_CONTROL();
\
                                                                            _tx_vfp_state =  _tx_vfp_state & ~
((ULONG) 0x4);                                          \
                                                                            __set_CONTROL(_tx_vfp_state);
\
                                                                        }
\
                                                                        else
\
                                                                        {
\
                                                                        ULONG  _tx_fpccr;
\
                                                                            _tx_fpccr =  *((ULONG *)
0xE000EF34);                                                 \
                                                                            _tx_fpccr =  _tx_fpccr & ((ULONG)
0x01);                                                  \
                                                                            if (_tx_fpccr == ((ULONG) 0x01))
\
                                                                            {
\
```

```
\
\
__get_CONTROL();                                                    \
& ((ULONG) 0x4);                                    \
s0");                                                \
0))                                                  \
\
__get_CONTROL();                                            \
_tx_vfp_state & ~((ULONG) 0x4);                             \
__set_CONTROL(_tx_vfp_state);                                       \
\
\
\
#else
#define TX_THREAD_TERMINATED_EXTENSION(thread_ptr)                  \
\
\
TX_THREAD_GET_SYSTEM_STATE();
&& ((thread_ptr) == _tx_thread_current_ptr))        \
\
\
_tx_misra_control_get();                                            \
((ULONG) 0x4);                                      \
_tx_misra_control_set(_tx_vfp_state);                               \
\
\
\
\
_tx_misra_fpccr_get();
0x01);                                              \
\
\
\
_tx_misra_control_get();                                            \
& ((ULONG) 0x4);                                    \
\
```

```
                ULONG _tx_vfp_state;
                    _tx_vfp_state =
            \
                    _tx_vfp_state =  _tx_vfp_state
                    __asm volatile ("vmov.f32 s0,
                    if (_tx_vfp_state == ((ULONG)
                    {
                        _tx_vfp_state =
                    _tx_vfp_state =
                  \
                    }
                }
            }
        }
    {
    ULONG  _tx_system_state;
        _tx_system_state =
      \
        if ((_tx_system_state == ((ULONG) 0))
        {
        ULONG  _tx_vfp_state;
            _tx_vfp_state =
     \
            _tx_vfp_state =  _tx_vfp_state & ~
                    \
        }
        else
        {
        ULONG  _tx_fpccr;
            _tx_fpccr =
         \
            _tx_fpccr =  _tx_fpccr & ((ULONG)
            if (_tx_fpccr == ((ULONG) 0x01))
            {
            ULONG _tx_vfp_state;
                _tx_vfp_state =
            \
                _tx_vfp_state =  _tx_vfp_state
                _tx_misra_vfp_touch();
```

```
\
                                                                        if (_tx_vfp_state == ((ULONG)
0))                                                 \
                                                                        {
\
                                                                            _tx_vfp_state =
_tx_misra_control_get();                                            \
                                                                            _tx_vfp_state =
_tx_vfp_state & ~((ULONG) 0x4);                                     \
_tx_misra_control_set(_tx_vfp_state);                                         \
                                                                        }
\
                                                                    }
\
                                                                }
\
                                                            }
#endif

#else

#define TX_THREAD_COMPLETED_EXTENSION(thread_ptr)
#define TX_THREAD_TERMINATED_EXTENSION(thread_ptr)

#endif


/* 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 the get system state macro.   */

#ifndef TX_THREAD_GET_SYSTEM_STATE
#ifndef TX_MISRA_ENABLE
#define TX_THREAD_GET_SYSTEM_STATE()        (_tx_thread_system_state | __get_IPSR())
#else
ULONG   _tx_misra_ipsr_get(VOID);
#define TX_THREAD_GET_SYSTEM_STATE()        (_tx_thread_system_state | _tx_misra_ipsr_get())
#endif
#endif


/* Define the check for whether or not to call the _tx_thread_system_return function.  A non-zero value
   indicates that _tx_thread_system_return should not be called. This overrides the definition in tx_thread.h
   for Cortex-M since so we don't waste time checking the _tx_thread_system_state variable that is always
   zero after initialization for Cortex-M ports. */

#ifndef TX_THREAD_SYSTEM_RETURN_CHECK
#define TX_THREAD_SYSTEM_RETURN_CHECK(c)    (c) = ((ULONG) _tx_thread_preempt_disable);
```

```
#endif


/* Define the macro to ensure _tx_thread_preempt_disable is set early in initialization in order to
   prevent early scheduling on Cortex-M parts.  */

#define TX_PORT_SPECIFIC_POST_INITIALIZATION    _tx_thread_preempt_disable++;


/* Determine if the ARM architecture has the CLZ instruction. This is available on
   architectures v5 and above. If available, redefine the macro for calculating the
   lowest bit set.  */

#ifndef TX_DISABLE_INLINE

#define TX_LOWEST_SET_BIT_CALCULATE(m, b)       (b) = (UINT)__CLZ(__RBIT((m)));

#endif


/* 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.  */

#ifdef TX_DISABLE_INLINE

UINT                                    _tx_thread_interrupt_disable(VOID);
VOID                                    _tx_thread_interrupt_restore(UINT previous_posture);

#define TX_INTERRUPT_SAVE_AREA                  register UINT interrupt_save;

#define TX_DISABLE                              interrupt_save = _tx_thread_interrupt_disable();

#define TX_RESTORE                              _tx_thread_interrupt_restore(interrupt_save);

#else

#define TX_INTERRUPT_SAVE_AREA                  __istate_t interrupt_save;
#define TX_DISABLE                              {interrupt_save =
__get_interrupt_state();__disable_interrupt();};
#define TX_RESTORE                              {__set_interrupt_state(interrupt_save);};

#define _tx_thread_system_return                _tx_thread_system_return_inline

static void _tx_thread_system_return_inline(void)
{
__istate_t interrupt_save;

    /* Set PendSV to invoke ThreadX scheduler.  */
    *((ULONG *) 0xE000ED04) = ((ULONG) 0x10000000);
    if (__get_IPSR() == 0)
    {
        interrupt_save = __get_interrupt_state();
        __enable_interrupt();
        __set_interrupt_state(interrupt_save);
    }
}

#endif


/* Define FPU extension for the Cortex-M4.  Each is assumed to be called in the context of the executing
   thread. These are no longer needed, but are preserved for backward compatibility only.  */

void    tx_thread_fpu_enable(void);
void    tx_thread_fpu_disable(void);
```

```
/* 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_FLAGS_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) Microsoft Corporation. All rights reserved.  *  ThreadX
Cortex-M4/IAR Version 6.0 *";
#else
#ifdef TX_MISRA_ENABLE
extern  CHAR                    _tx_version_id[100];
#else
extern  CHAR                    _tx_version_id[];
#endif
#endif


#endif
```

Appendix B - The tx_trace.h file

5/21/2020 • 16 minutes to read

This chapter displays the **tx_trace.h** file.

```
/**************************************************************************/
/*                                                                        */
/*       Copyright (c) Microsoft Corporation. All rights reserved.        */
/*                                                                        */
/*       This software is licensed under the Microsoft Software License   */
/*       Terms for Microsoft Azure RTOS. Full text of the license can be  */
/*       found in the LICENSE file at https://aka.ms/AzureRTOS_EULA       */
/*       and in the root directory of this software.                      */
/*                                                                        */
/**************************************************************************/


/**************************************************************************/
/**************************************************************************/
/**                                                                       */
/** ThreadX Component                                                     */
/**                                                                       */
/**   Trace                                                               */
/**                                                                       */
/**************************************************************************/
/**************************************************************************/

/**************************************************************************/
/*                                                                        */
/*  COMPONENT DEFINITION                                   RELEASE        */
/*                                                                        */
/*    tx_trace.h                                          PORTABLE C      */
/*                                                           6.0          */
/*  AUTHOR                                                                */
/*                                                                        */
/*    William E. Lamie, Microsoft Corporation                             */
/*                                                                        */
/*  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             */
/*                                                                        */
/*  05-19-2020     William E. Lamie         Initial Version 6.0           */
/*                                                                        */
/**************************************************************************/


/* 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_INITIALIZE
#define TX_TRACE_OBJECT_REGISTER(t,p,n,a,b)
#define TX_TRACE_OBJECT_UNREGISTER(o)
#define TX_TRACE_IN_LINE_INSERT(i,a,b,c,d,f)
#else

/* Event tracing is enabled.  */

/* Ensure that the thread component information is included.  */

#include "tx_thread.h"


/* Define trace port-specfic 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 analyzed 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 (TXTB). 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
            [Reserved]                          2   Reserved two bytes - should be 0x0000
            [Trace Object Object Name Size]     2   Number of bytes in object name
            [Trace Object Registry End Pointer] 4   Pointer to the end of Trace Object Registry
            [Trace Buffer Start Pointer]        4   Pointer to the start of the Trace Buffer Area
            [Trace Buffer End Pointer]          4   Pointer to the end of the 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).
            [Reserved]                          4   Reserved 4 bytes, should be 0xAAAAAAAA
            [Reserved]                          4   Reserved 4 bytes, should be 0xBBBBBBBB
            [Reserved]                          4   Reserved 4 bytes, should be 0xCCCCCCCC
*/


/* Define the Trace Control Header.  */

typedef struct TX_TRACE_HEADER_STRUCT
{

    ULONG                                           tx_trace_header_id;
    ULONG                                           tx_trace_header_timer_valid_mask;
    ULONG                                           tx_trace_header_trace_base_address;
    ULONG                                           tx_trace_header_registry_start_pointer;
    USHORT                                          tx_trace_header_reserved1;
    USHORT                                          tx_trace_header_object_name_size;
    ULONG                                           tx_trace_header_registry_end_pointer;
    ULONG                                           tx_trace_header_buffer_start_pointer;
    ULONG                                           tx_trace_header_buffer_end_pointer;
    ULONG                                           tx_trace_header_buffer_current_pointer;
    ULONG                                           tx_trace_header_reserved2;
    ULONG                                           tx_trace_header_reserved3;
    ULONG                                           tx_trace_header_reserved4;
} TX_TRACE_HEADER;


/* Trace Object Registry. The Trace Object Registry is used to map the object pointer in the trace buffer to
   the application's name for the object (defined during object creation in ThreadX).  */

#ifndef TX_TRACE_OBJECT_REGISTRY_NAME
#define TX_TRACE_OBJECT_REGISTRY_NAME                   32
#endif


/* Define the object name types as well as the contents of any additional parameters that might be useful in
   trace analysis.  */

#define TX_TRACE_OBJECT_TYPE_NOT_VALID                  ((UCHAR) 0)     /* Object is not valid
*/
#define TX_TRACE_OBJECT_TYPE_THREAD                     ((UCHAR) 1)     /* P1 = stack start address, P2 =
stack size         */
#define TX_TRACE_OBJECT_TYPE_TIMER                      ((UCHAR) 2)     /* P1 = initial ticks, P2 =
reschedule ticks         */
#define TX_TRACE_OBJECT_TYPE_QUEUE                      ((UCHAR) 3)     /* P1 = queue size, P2 = message
size               */
```

```
#define TX_TRACE_OBJECT_TYPE_SEMAPHORE                      ((UCHAR) 4)     /* P1 = initial instances
*/
#define TX_TRACE_OBJECT_TYPE_MUTEX                          ((UCHAR) 5)     /* P1 = priority inheritance flag
*/
#define TX_TRACE_OBJECT_TYPE_EVENT_FLAGS                    ((UCHAR) 6)     /* none
*/
#define TX_TRACE_OBJECT_TYPE_BLOCK_POOL                     ((UCHAR) 7)     /* P1 = total blocks, P2 = block
size                */
#define TX_TRACE_OBJECT_TYPE_BYTE_POOL                      ((UCHAR) 8)     /* P1 = total bytes
*/


typedef struct TX_TRACE_OBJECT_ENTRY_STRUCT
{

    UCHAR                                               tx_trace_object_entry_available;
/* TX_TRUE -> available                 */
    UCHAR                                               tx_trace_object_entry_type;
/* Types defined above                  */
    UCHAR                                               tx_trace_object_entry_reserved1;
/* Should be zero - except for thread   */
    UCHAR                                               tx_trace_object_entry_reserved2;
/* Should be zero - except for thread   */
    ULONG                                               tx_trace_object_entry_thread_pointer;
/* ThreadX object pointer               */
    ULONG                                               tx_trace_object_entry_param_1;
/* Parameter value defined              */
    ULONG                                               tx_trace_object_entry_param_2;
/*   according to type above            */
    UCHAR
tx_trace_object_entry_name[TX_TRACE_OBJECT_REGISTRY_NAME];  /* Object name                          */
} TX_TRACE_OBJECT_ENTRY;


/* 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:

                Entry                   Size                    Description

            [Thread Pointer]            4           This 4-byte field contains the pointer to the
                                                    ThreadX thread running that caused the event.
                                                    If this field is NULL, the entry hasn't been used
                                                    yet. If this field is 0xFFFFFFFF, the event occurred
                                                    from within an ISR. If this entry is 0xF0F0F0F0, the
                                                    event occurred during initialization.
            [Thread Priority or         4           This 4-byte field contains the current thread pointer for
interrupt
             Current Thread                         events or the thread preemption-threshold/priority for
thread events.
             Preemption-Threshold/
             Priority]
            [Event ID]                  4           This 4-byte field contains the Event ID of the event. A
value of
                                                    0xFFFFFFFF indicates the event is invalid. All events are
marked
                                                    as invalid during initialization.
            [Time Stamp]                4           This 4-byte field contains the time stamp of the event.
            [Information Field 1]       4           This 4-byte field contains the first 4-bytes of
information
                                                    specific to the event.
            [Information Field 2]       4           This 4-byte field contains the second 4-bytes of
information
                                                    specific to the event.
            [Information Field 3]       4           This 4-byte field contains the third 4-bytes of
information
                                                    specific to the event.
            [Information Field 4]       4           This 4-byte field contains the fourth 4-bytes of
information
                                                    specific to the event.
```

```
                                                specific to the event.
*/

#define TX_TRACE_INVALID_EVENT                              0xFFFFFFFFUL


/* Define ThreadX Trace Events, along with a brief description of the additional information fields,
   where I1 -> Information Field 1, I2 -> Information Field 2, etc.  */

/* Event numbers 0 through 4095 are reserved by Azure RTOS. Specific event assignments are:

                                ThreadX events:     1-199
                                FileX events:       200-299
                                NetX events:        300-599
                                USBX events:        600-999

   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.  */

/* Define the basic ThreadX thread scheduling events first.  */

#define TX_TRACE_THREAD_RESUME                              1           /* I1 = thread ptr, I2 =
previous_state, I3 = stack ptr, I4 = next thread   */
#define TX_TRACE_THREAD_SUSPEND                             2           /* I1 = thread ptr, I2 = new_state, I3
= stack ptr  I4 = next thread        */
#define TX_TRACE_ISR_ENTER                                  3           /* I1 = stack_ptr, I2 = ISR number, I3
= system state, I4 = preempt disable */
#define TX_TRACE_ISR_EXIT                                   4           /* I1 = stack_ptr, I2 = ISR number, I3
= system state, I4 = preempt disable */
#define TX_TRACE_TIME_SLICE                                 5           /* I1 = next thread ptr, I2 = system
state, I3 = preempt disable, I4 = stack*/
#define TX_TRACE_RUNNING                                    6           /* None
*/


/* Define the rest of the ThreadX system events.  */

#define TX_TRACE_BLOCK_ALLOCATE                             10          /* I1 = pool ptr, I2 = memory ptr, I3
= wait option, I4 = remaining blocks  */
#define TX_TRACE_BLOCK_POOL_CREATE                          11          /* I1 = pool ptr, I2 = pool_start, I3
= total blocks, I4 = block size       */
#define TX_TRACE_BLOCK_POOL_DELETE                          12          /* I1 = pool ptr, I2 = stack ptr
*/
#define TX_TRACE_BLOCK_POOL_INFO_GET                        13          /* I1 = pool ptr
*/
#define TX_TRACE_BLOCK_POOL_PERFORMANCE_INFO_GET            14          /* I1 = pool ptr
*/
#define TX_TRACE_BLOCK_POOL__PERFORMANCE_SYSTEM_INFO_GET    15          /* None
*/
#define TX_TRACE_BLOCK_POOL_PRIORITIZE                      16          /* I1 = pool ptr, I2 = suspended
count, I3 = stack ptr                       */
#define TX_TRACE_BLOCK_RELEASE                              17          /* I1 = pool ptr, I2 = memory ptr, I3
= suspended, I4 = stack ptr           */
#define TX_TRACE_BYTE_ALLOCATE                              20          /* I1 = pool ptr, I2 = memory ptr, I3
= size requested, I4 = wait option    */
#define TX_TRACE_BYTE_POOL_CREATE                           21          /* I1 = pool ptr, I2 = start ptr, I3 =
pool size, I4 = stack ptr            */
#define TX_TRACE_BYTE_POOL_DELETE                           22          /* I1 = pool ptr, I2 = stack ptr
*/
#define TX_TRACE_BYTE_POOL_INFO_GET                         23          /* I1 = pool ptr
*/
#define TX_TRACE_BYTE_POOL_PERFORMANCE_INFO_GET             24          /* I1 = pool ptr
*/
#define TX_TRACE_BYTE_POOL__PERFORMANCE_SYSTEM_INFO_GET     25          /* None
*/
#define TX_TRACE_BYTE_POOL_PRIORITIZE                       26          /* I1 = pool ptr, I2 = suspended
count, I3 = stack ptr                       */
#define TX_TRACE_BYTE_RELEASE                               27          /* I1 = pool ptr, I2 = memory ptr, I3
= suspended, I4 = available bytes     */
```

```
= suspended, I4 = available bytes      */
#define TX_TRACE_EVENT_FLAGS_CREATE                         30          /* I1 = group ptr, I2 = stack ptr
*/
#define TX_TRACE_EVENT_FLAGS_DELETE                         31          /* I1 = group ptr, I2 = stack ptr
*/
#define TX_TRACE_EVENT_FLAGS_GET                            32          /* I1 = group ptr, I2 = requested
flags, I3 = current flags, I4 = get option*/
#define TX_TRACE_EVENT_FLAGS_INFO_GET                       33          /* I1 = group ptr
*/
#define TX_TRACE_EVENT_FLAGS_PERFORMANCE_INFO_GET           34          /* I1 = group ptr
*/
#define TX_TRACE_EVENT_FLAGS__PERFORMANCE_SYSTEM_INFO_GET   35          /* None
*/
#define TX_TRACE_EVENT_FLAGS_SET                            36          /* I1 = group ptr, I2 = flags to set,
I3 = set option, I4= suspended count  */
#define TX_TRACE_EVENT_FLAGS_SET_NOTIFY                     37          /* I1 = group ptr
*/
#define TX_TRACE_INTERRUPT_CONTROL                          40          /* I1 = new interrupt posture, I2 =
stack ptr                               */
#define TX_TRACE_MUTEX_CREATE                               50          /* I1 = mutex ptr, I2 = inheritance,
I3 = stack ptr                          */
#define TX_TRACE_MUTEX_DELETE                               51          /* I1 = mutex ptr, I2 = stack ptr
*/
#define TX_TRACE_MUTEX_GET                                  52          /* I1 = mutex ptr, I2 = wait option,
I3 = owning thread, I4 = own count     */
#define TX_TRACE_MUTEX_INFO_GET                             53          /* I1 = mutex ptr
*/
#define TX_TRACE_MUTEX_PERFORMANCE_INFO_GET                 54          /* I1 = mutex ptr
*/
#define TX_TRACE_MUTEX_PERFORMANCE_SYSTEM_INFO_GET          55          /* None
*/
#define TX_TRACE_MUTEX_PRIORITIZE                           56          /* I1 = mutex 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_QUEUE_CREATE                               60          /* I1 = queue ptr, I2 = message size,
I3 = queue start, I4 = queue size     */
#define TX_TRACE_QUEUE_DELETE                               61          /* I1 = queue ptr, I2 = stack ptr
*/
#define TX_TRACE_QUEUE_FLUSH                                62          /* I1 = queue ptr, I2 = stack ptr
*/
#define TX_TRACE_QUEUE_FRONT_SEND                           63          /* I1 = queue ptr, I2 = source ptr, I3
= wait option, I4 = enqueued         */
#define TX_TRACE_QUEUE_INFO_GET                             64          /* I1 = queue ptr
*/
#define TX_TRACE_QUEUE_PERFORMANCE_INFO_GET                 65          /* I1 = queue ptr
*/
#define TX_TRACE_QUEUE_PERFORMANCE_SYSTEM_INFO_GET          66          /* None
*/
#define TX_TRACE_QUEUE_PRIORITIZE                           67          /* I1 = queue ptr, I2 = suspended
count, I3 = stack ptr                      */
#define TX_TRACE_QUEUE_RECEIVE                              68          /* I1 = queue ptr, I2 = destination
ptr, I3 = wait option, I4 = enqueued    */
#define TX_TRACE_QUEUE_SEND                                 69          /* I1 = queue ptr, I2 = source ptr, I3
= wait option, I4 = enqueued         */
#define TX_TRACE_QUEUE_SEND_NOTIFY                          70          /* I1 = queue ptr
*/
#define TX_TRACE_SEMAPHORE_CEILING_PUT                      80          /* I1 = semaphore ptr, I2 = current
count, I3 = suspended count,I4 =ceiling */
#define TX_TRACE_SEMAPHORE_CREATE                           81          /* I1 = semaphore ptr, I2 = initial
count, I3 = stack ptr                    */
#define TX_TRACE_SEMAPHORE_DELETE                           82          /* I1 = semaphore ptr, I2 = stack ptr
*/
#define TX_TRACE_SEMAPHORE_GET                              83          /* I1 = semaphore ptr, I2 = wait
option, I3 = current count, I4 = stack ptr */
#define TX_TRACE_SEMAPHORE_INFO_GET                         84          /* I1 = semaphore ptr
*/
#define TX_TRACE_SEMAPHORE_PERFORMANCE_INFO_GET             85          /* I1 = semaphore ptr
*/
#define TX_TRACE_SEMAPHORE__PERFORMANCE_SYSTEM_INFO_GET     86          /* None
```

```
#define TX_TRACE_SEMAPHORE__PERFORMANCE_SYSTEM_INFO_GET     86          /* None
*/
#define TX_TRACE_SEMAPHORE_PRIORITIZE                       87          /* I1 = semaphore ptr, I2 = suspended
count, I2 = stack ptr                */
#define TX_TRACE_SEMAPHORE_PUT                              88          /* I1 = semaphore ptr, I2 = current
count, I3 = suspended count,I4=stack ptr*/
#define TX_TRACE_SEMAPHORE_PUT_NOTIFY                       89          /* I1 = semaphore ptr
*/
#define TX_TRACE_THREAD_CREATE                              100         /* I1 = thread ptr, I2 = priority, I3
= stack ptr, I4 = stack_size          */
#define TX_TRACE_THREAD_DELETE                              101         /* I1 = thread ptr, I2 = stack ptr
*/
#define TX_TRACE_THREAD_ENTRY_EXIT_NOTIFY                   102         /* I1 = thread ptr, I2 = thread state,
I3 = stack ptr                       */
#define TX_TRACE_THREAD_IDENTIFY                            103         /* None
*/
#define TX_TRACE_THREAD_INFO_GET                            104         /* I1 = thread ptr, I2 = thread state
*/
#define TX_TRACE_THREAD_PERFORMANCE_INFO_GET                105         /* I1 = thread ptr, I2 = thread state
*/
#define TX_TRACE_THREAD_PERFORMANCE_SYSTEM_INFO_GET         106         /* None
*/
#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                     108         /* I1 = thread ptr, I2 = new priority,
I3 = old priority, I4 = thread state */
#define TX_TRACE_THREAD_RELINQUISH                          109         /* I1 = stack ptr, I2 = next thread
ptr                                  */
#define TX_TRACE_THREAD_RESET                               110         /* I1 = thread ptr, I2 = thread state
*/
#define TX_TRACE_THREAD_RESUME_API                          111         /* I1 = thread ptr, I2 = thread state,
I3 = stack ptr                       */
#define TX_TRACE_THREAD_SLEEP                               112         /* I1 = sleep value, I2 = thread
state, I3 = stack ptr                     */
#define TX_TRACE_THREAD_STACK_ERROR_NOTIFY                  113         /* None
*/
#define TX_TRACE_THREAD_SUSPEND_API                         114         /* I1 = thread ptr, I2 = thread state,
I3 = stack ptr                       */
#define TX_TRACE_THREAD_TERMINATE                           115         /* I1 = thread ptr, I2 = thread state,
I3 = stack ptr                       */
#define TX_TRACE_THREAD_TIME_SLICE_CHANGE                   116         /* I1 = thread ptr, I2 = new
timeslice, I3 = old timeslice                 */
#define TX_TRACE_THREAD_WAIT_ABORT                          117         /* I1 = thread ptr, I2 = thread state,
I3 = stack ptr                       */
#define TX_TRACE_TIME_GET                                   120         /* I1 = current time, I2 = stack ptr
*/
#define TX_TRACE_TIME_SET                                   121         /* I1 = new time
*/
#define TX_TRACE_TIMER_ACTIVATE                             122         /* I1 = timer ptr
*/
#define TX_TRACE_TIMER_CHANGE                               123         /* I1 = timer ptr, I2 = initial ticks,
I3= reschedule ticks                 */
#define TX_TRACE_TIMER_CREATE                               124         /* I1 = timer ptr, I2 = initial ticks,
I3= reschedule ticks, I4 = enable    */
#define TX_TRACE_TIMER_DEACTIVATE                           125         /* I1 = timer ptr, I2 = stack ptr
*/
#define TX_TRACE_TIMER_DELETE                               126         /* I1 = timer ptr
*/
#define TX_TRACE_TIMER_INFO_GET                             127         /* I1 = timer ptr, I2 = stack ptr
*/
#define TX_TRACE_TIMER_PERFORMANCE_INFO_GET                 128         /* I1 = timer ptr
*/
#define TX_TRACE_TIMER_PERFORMANCE_SYSTEM_INFO_GET          129         /* None
*/


/* Define the an Trace Buffer Entry.  */

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;
#ifdef TX_MISRA_ENABLE
    ULONG                                               tx_trace_buffer_entry_info_1;
    ULONG                                               tx_trace_buffer_entry_info_2;
    ULONG                                               tx_trace_buffer_entry_info_3;
    ULONG                                               tx_trace_buffer_entry_info_4;
#else
    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;
#endif
} 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
#else
#define TRACE_DECLARE extern
#endif


/* Define the pointer to the start of the trace buffer control structure.   */

TRACE_DECLARE  TX_TRACE_HEADER                  *_tx_trace_header_ptr;


/* Define the pointer to the start of the trace object registry area in the trace buffer.  */

TRACE_DECLARE  TX_TRACE_OBJECT_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_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 *buffer);


/* Define the total number of registry entries.  */

TRACE_DECLARE  ULONG                            _tx_trace_total_registry_entries;


/* Define a counter that is used to track the number of available registry entries.  */

TRACE_DECLARE  ULONG                            _tx_trace_available_registry_entries;


/* Define an index that represents the start of the registry search.  */

TRACE_DECLARE  ULONG                            _tx_trace_registry_search_start;


/* Define the event trace macros that are expanded in-line when event tracing is enabled.  */

#ifdef TX_MISRA_ENABLE
#define TX_TRACE_INFO_FIELD_ASSIGNMENT(a,b,c,d)  trace_event_ptr -> tx_trace_buffer_entry_info_1 =  (ULONG)
(a); trace_event_ptr -> tx_trace_buffer_entry_info_2 =  (ULONG) (b); trace_event_ptr ->
tx_trace_buffer_entry_info_3 =  (ULONG) (c); trace_event_ptr -> tx_trace_buffer_entry_info_4 =  (ULONG) (d);
#else
#define TX_TRACE_INFO_FIELD_ASSIGNMENT(a,b,c,d)  trace_event_ptr -> tx_trace_buffer_entry_information_field_1
=  (ULONG) (a); trace_event_ptr -> tx_trace_buffer_entry_information_field_2 =  (ULONG) (b); trace_event_ptr -
> tx_trace_buffer_entry_information_field_3 =  (ULONG) (c); trace_event_ptr ->
tx_trace_buffer_entry_information_field_4 =  (ULONG) (d);
#endif


#define TX_TRACE_INITIALIZE                                     _tx_trace_initialize();
#define TX_TRACE_OBJECT_REGISTER(t,p,n,a,b)                     _tx_trace_object_register((UCHAR) (t), (VOID
*) (p), (CHAR *) (n), (ULONG) (a), (ULONG) (b));
#define TX_TRACE_OBJECT_UNREGISTER(o)                           _tx_trace_object_unregister((VOID *) (o));
#ifndef TX_TRACE_IN_LINE_INSERT
#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_GET_SYSTEM_STATE(); \
                TX_THREAD_GET_CURRENT(trace_thread_ptr) \
                \
                if (trace_system_state == 0) \
                { \
                    trace_priority =  trace_thread_ptr -> tx_thread_priority; \
                    trace_priority =  trace_priority | 0x80000000UL | (trace_thread_ptr ->
tx_thread_preempt_threshold << 16); \
                } \
                else if (trace_system_state < 0xF0F0F0F0UL) \
                { \
                    trace_priority =    (ULONG) trace_thread_ptr; \
                    trace_thread_ptr =  (TX_THREAD *) 0xFFFFFFFFUL; \
```

```
                } \
                else \
                { \
                    trace_thread_ptr =  (TX_THREAD *) 0xF0F0F0F0UL; \
                    trace_priority =    0; \
                } \
                trace_event_ptr -> tx_trace_buffer_entry_thread_pointer =       (ULONG) trace_thread_ptr; \
                trace_event_ptr -> tx_trace_buffer_entry_thread_priority =      (ULONG) trace_priority; \
                trace_event_ptr -> tx_trace_buffer_entry_event_id =             (ULONG) (i); \
                trace_event_ptr -> tx_trace_buffer_entry_time_stamp =           (ULONG) TX_TRACE_TIME_SOURCE;
\
                TX_TRACE_INFO_FIELD_ASSIGNMENT((a),(b),(c),(d)) \
                trace_event_ptr++; \
                if (trace_event_ptr >= _tx_trace_buffer_end_ptr) \
                { \
                    trace_event_ptr =  _tx_trace_buffer_start_ptr; \
                    _tx_trace_buffer_current_ptr =  trace_event_ptr;  \
                    _tx_trace_header_ptr -> tx_trace_header_buffer_current_pointer =  (ULONG) trace_event_ptr;
\
                    if (_tx_trace_full_notify_function) \
                        (_tx_trace_full_notify_function)((VOID *) _tx_trace_header_ptr); \
                } \
                else \
                { \
                    _tx_trace_buffer_current_ptr =  trace_event_ptr;  \
                    _tx_trace_header_ptr -> tx_trace_header_buffer_current_pointer =  (ULONG) trace_event_ptr;
\
                } \
            } \
        }
#endif
#endif


#ifdef TX_SOURCE_CODE

/* Define internal function prototypes of the trace component, only if compiling ThreadX source code.  */

VOID    _tx_trace_initialize(VOID);
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);


#ifdef TX_ENABLE_EVENT_TRACE

/* Check for MISRA compliance requirements.  */

#ifdef TX_MISRA_ENABLE

/* Define MISRA-specific routines.  */

UCHAR                   *_tx_misra_object_to_uchar_pointer_convert(TX_TRACE_OBJECT_ENTRY *pointer);
TX_TRACE_OBJECT_ENTRY   *_tx_misra_uchar_to_object_pointer_convert(UCHAR *pointer);
TX_TRACE_HEADER         *_tx_misra_uchar_to_header_pointer_convert(UCHAR *pointer);
TX_TRACE_BUFFER_ENTRY   *_tx_misra_uchar_to_entry_pointer_convert(UCHAR *pointer);
UCHAR                   *_tx_misra_entry_to_uchar_pointer_convert(TX_TRACE_BUFFER_ENTRY *pointer);


#define TX_OBJECT_TO_UCHAR_POINTER_CONVERT(a)           _tx_misra_object_to_uchar_pointer_convert((a))
#define TX_UCHAR_TO_OBJECT_POINTER_CONVERT(a)           _tx_misra_uchar_to_object_pointer_convert((a))
#define TX_UCHAR_TO_HEADER_POINTER_CONVERT(a)           _tx_misra_uchar_to_header_pointer_convert((a))
#define TX_UCHAR_TO_ENTRY_POINTER_CONVERT(a)            _tx_misra_uchar_to_entry_pointer_convert((a))
#define TX_ENTRY_TO_UCHAR_POINTER_CONVERT(a)            _tx_misra_entry_to_uchar_pointer_convert((a))

#else

#define TX_OBJECT_TO_UCHAR_POINTER_CONVERT(a)           ((UCHAR *) ((VOID *) (a)))
#define TX_UCHAR_TO_OBJECT_POINTER_CONVERT(a)           ((TX_TRACE_OBJECT_ENTRY *) ((VOID *) (a)))
```

```
#define TX_UCHAR_TO_HEADER_POINTER_CONVERT(a)           ((TX_TRACE_HEADER *) ((VOID *) (a)))
#define TX_UCHAR_TO_ENTRY_POINTER_CONVERT(a)           ((TX_TRACE_BUFFER_ENTRY *) ((VOID *) (a)))
#define TX_ENTRY_TO_UCHAR_POINTER_CONVERT(a)           ((UCHAR *) ((VOID *) (a)))

#endif
#endif
#endif
#endif
```

Appendix C - DOS command-line utilities

5/21/2020 • 2 minutes to read

There are three DOS command-line utilities found in the TraceX installation under the ***Utilities*** subdirectory.

The utilities supplied are listed below:

UTILITY	PURPOSE	COMMAND-LINE DEFINITIONS
ea2tracex.exe	Converts the trace file ea2tracex generated by ThreadX in original_file association with the GHS tools converted_file to the TraceX trace file format. The ThreadX for GHS tools produces a different trace format than ThreadX for non-GHS tools, which is why this conversion utility is needed.	`> eatracex original_file converted_file <cr>`
hex2tracex.exe	Converts a trace file generated by ThreadX but dumped from the development tool in Intel HEX format to the binary TraceX trace file format. TraceX V5 and above can open HEX files without converting them.	`hex2tracex hex_file converted_file <cr>`
mot2tracex.exe	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. TraceX V5 and above can open S-Record files without converting them.	`> mot2tracex mot_file converted_file <cr>`

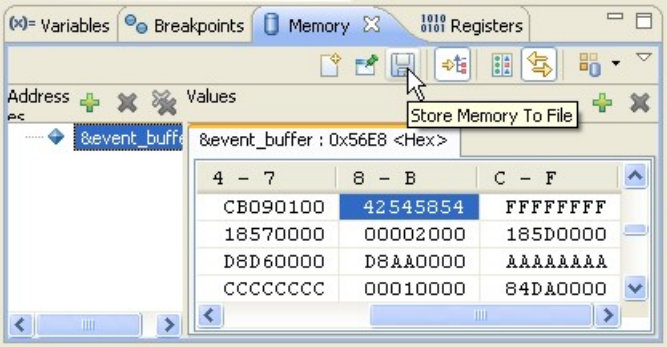
Appendix D - Dumping and trace buffer

7/20/2020 • 3 minutes to read

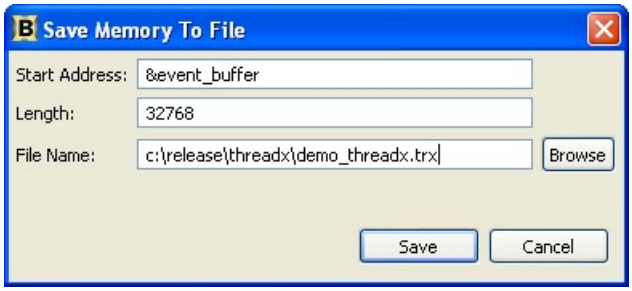
Dumping the trace buffer created by Azure RTOS ThreadX to a file on the host computer is done via commands and/or utilities provided by the specific development tool being used. This appendix contains examples of dumping a trace buffer to a host file in some of the more popular development tools used with ThreadX.

BenchX Tools

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

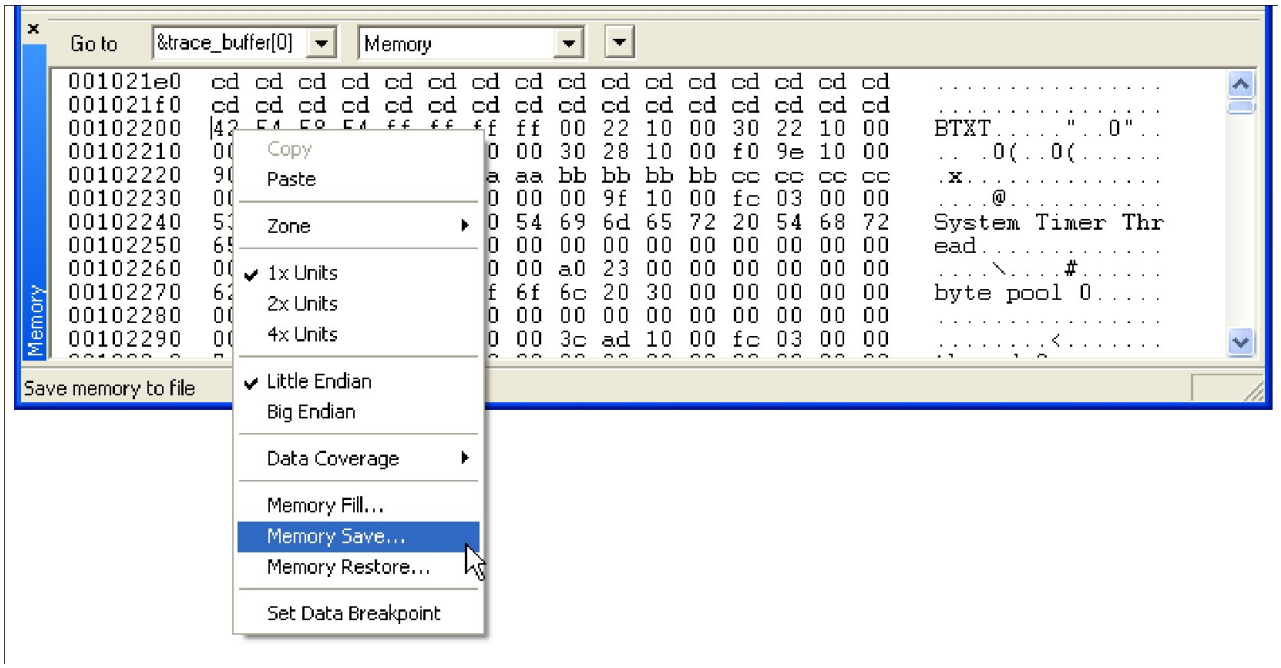
The trace buffer can be dumped to a host file easily with the ARM RealView tools by entering the following command at the command-line prompt in RealView:

```
> WRITEFILE,raw trace_file.trx=0x6860..0xE560
```

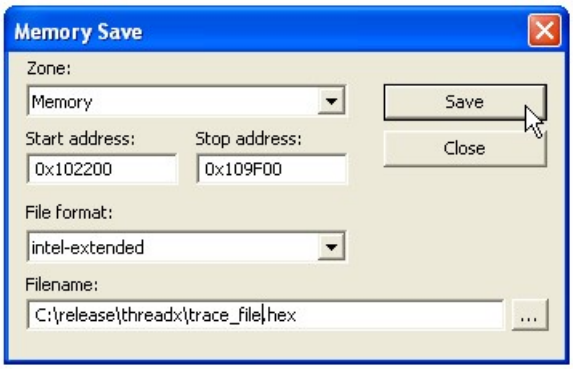
Upon completion, the file ***trace_file.trx*** will contain the trace buffer that is located starting at the address 0x6860 and goes up to address 0xE560. This file is ready for viewing by TraceX.

IAR Tools

The trace buffer can be dumped to a host file 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***.



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:

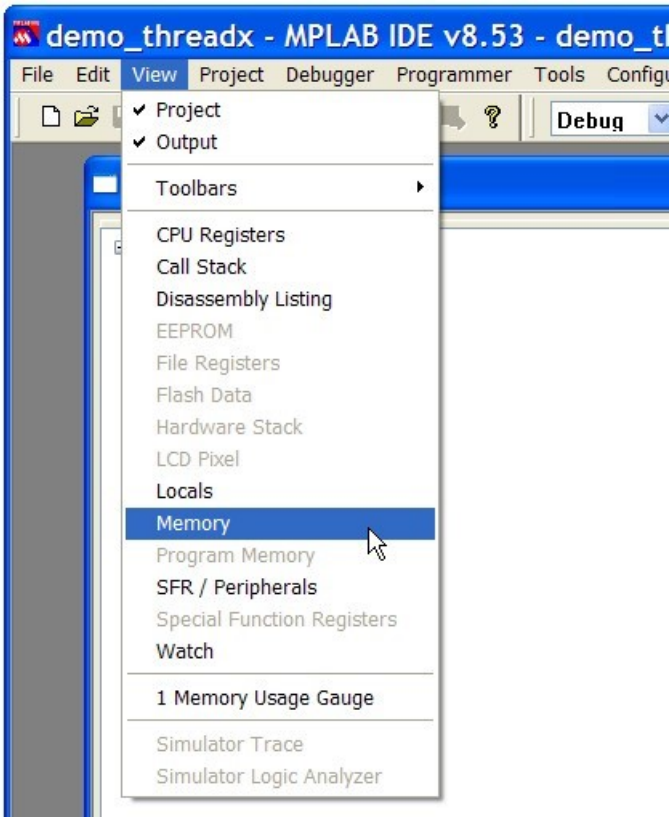
```
> save -b p:0x102200..0x109F00 trace_file.trx -a 32bit
```

This results in the trace buffer being saved in the file ***trace_file.trx*** on the host.

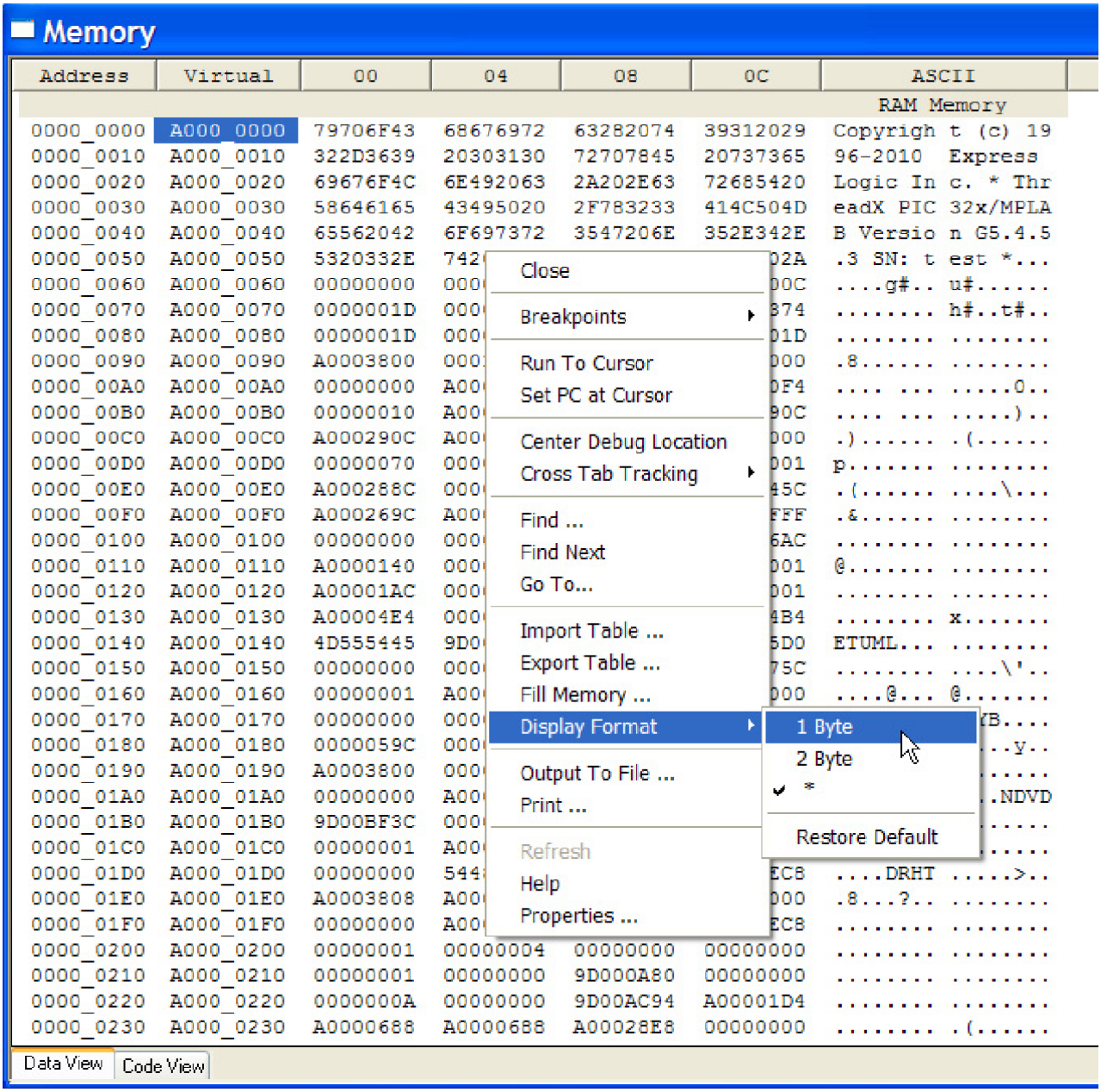
MPLAB Tools

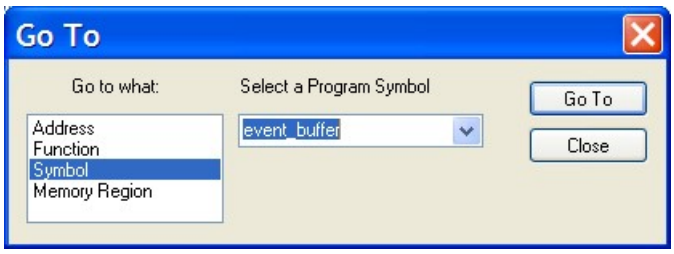
MPLAB can create a TraceX-compatible trace file through its Export Table utility, which allows the export of any range of memory to a host file. To use this utility to create a trace file for TraceX, proceed as follows:

Step 1 Open a memory window by selecting View -> Memory.

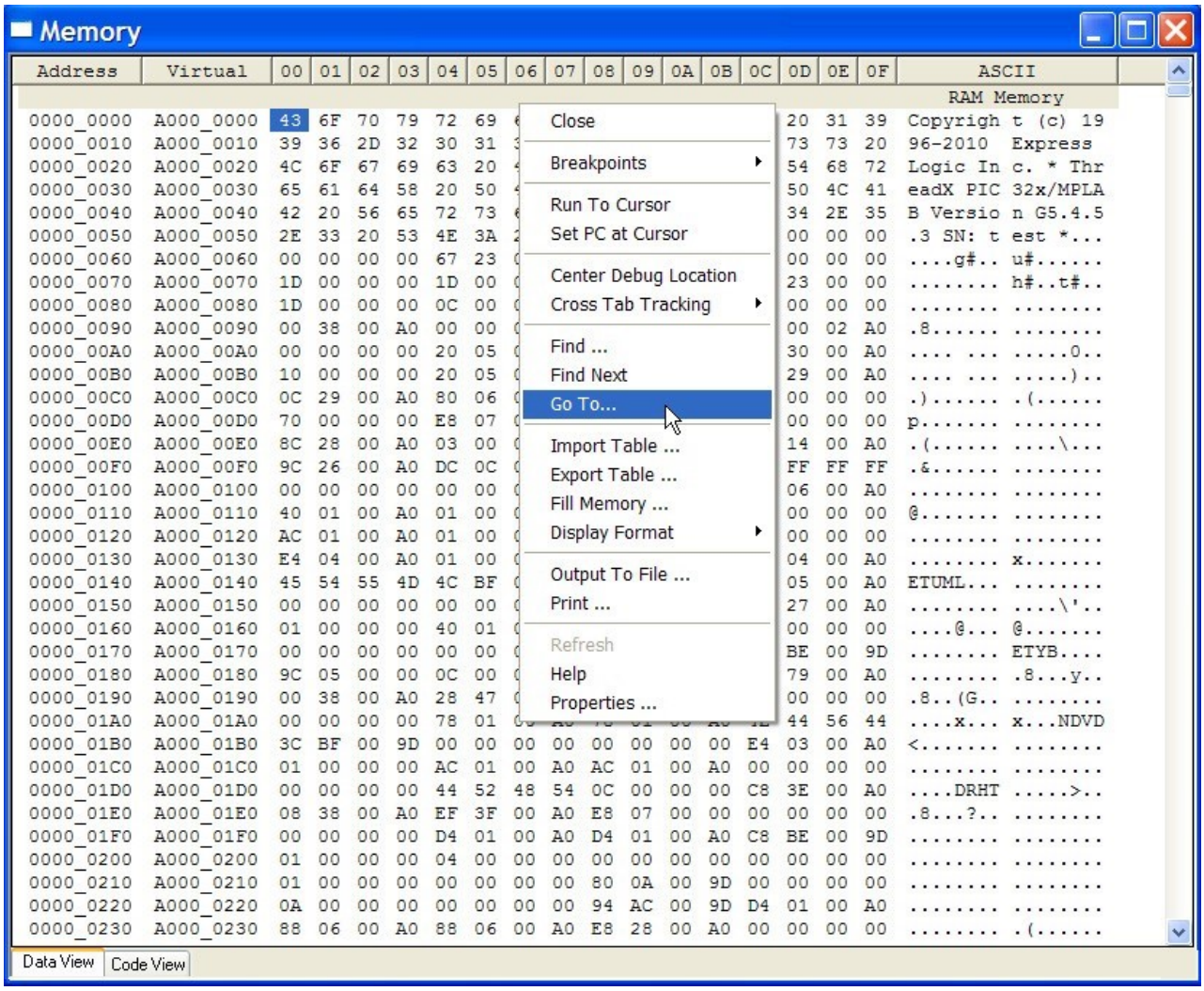


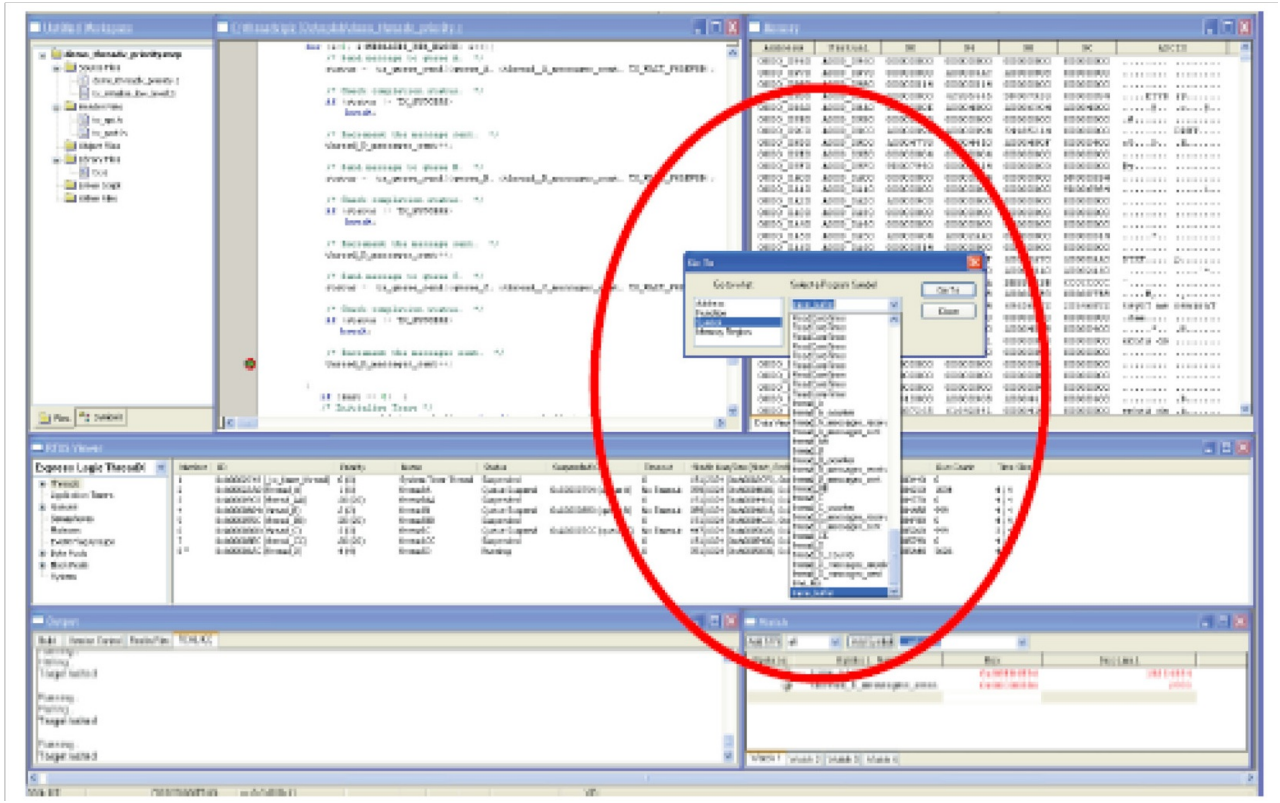
Step 2 Right-click within the **Memory View** to display a list of options. Specify **Display Format -1 Byte** to select byte display..



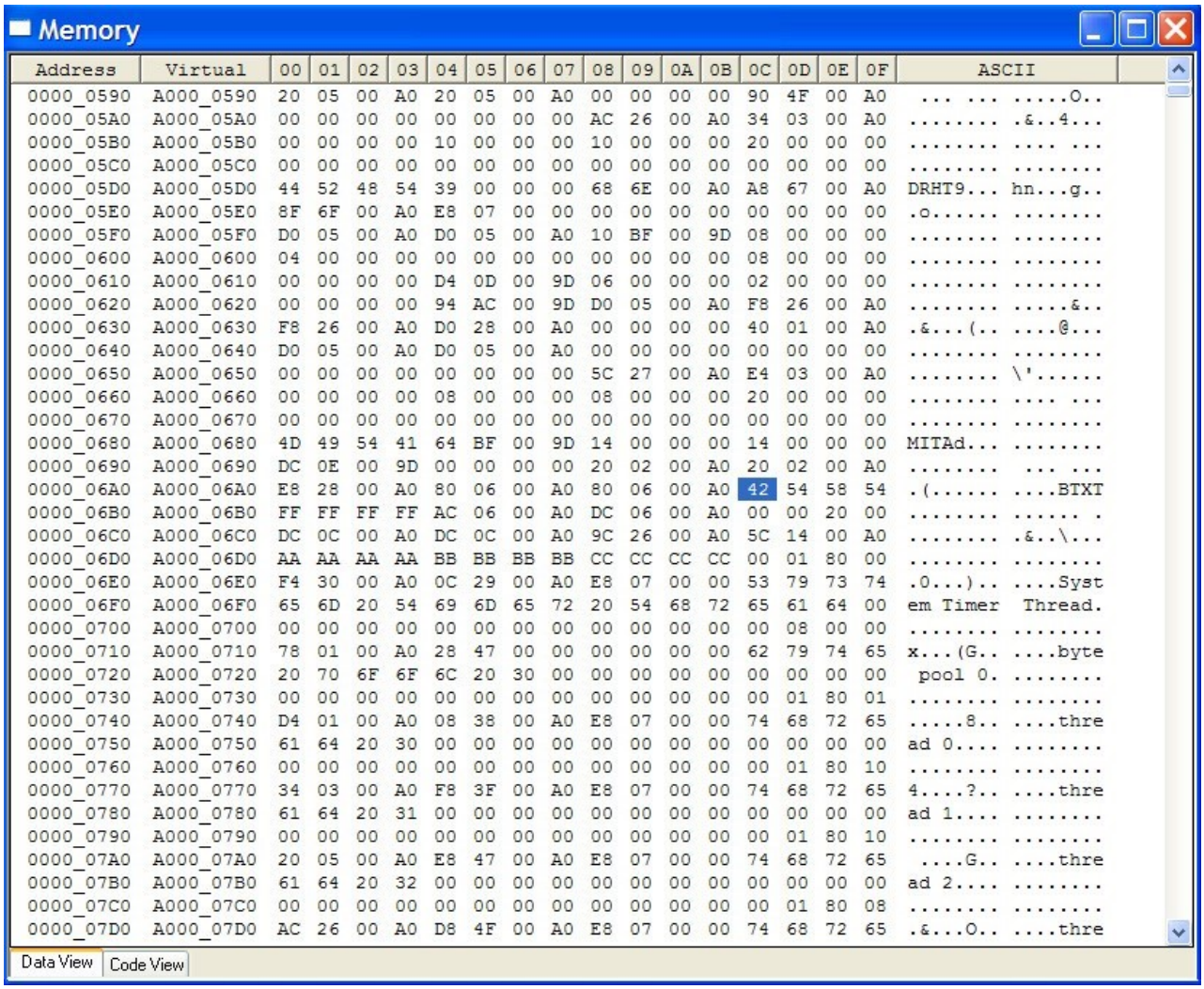


Step 3 Right-click again within the **Memory View** Window and select **Go To**, which opens a dialog box that enables you to specify the address of the event buffer. This example shows ***event_buffer*** being displayed.

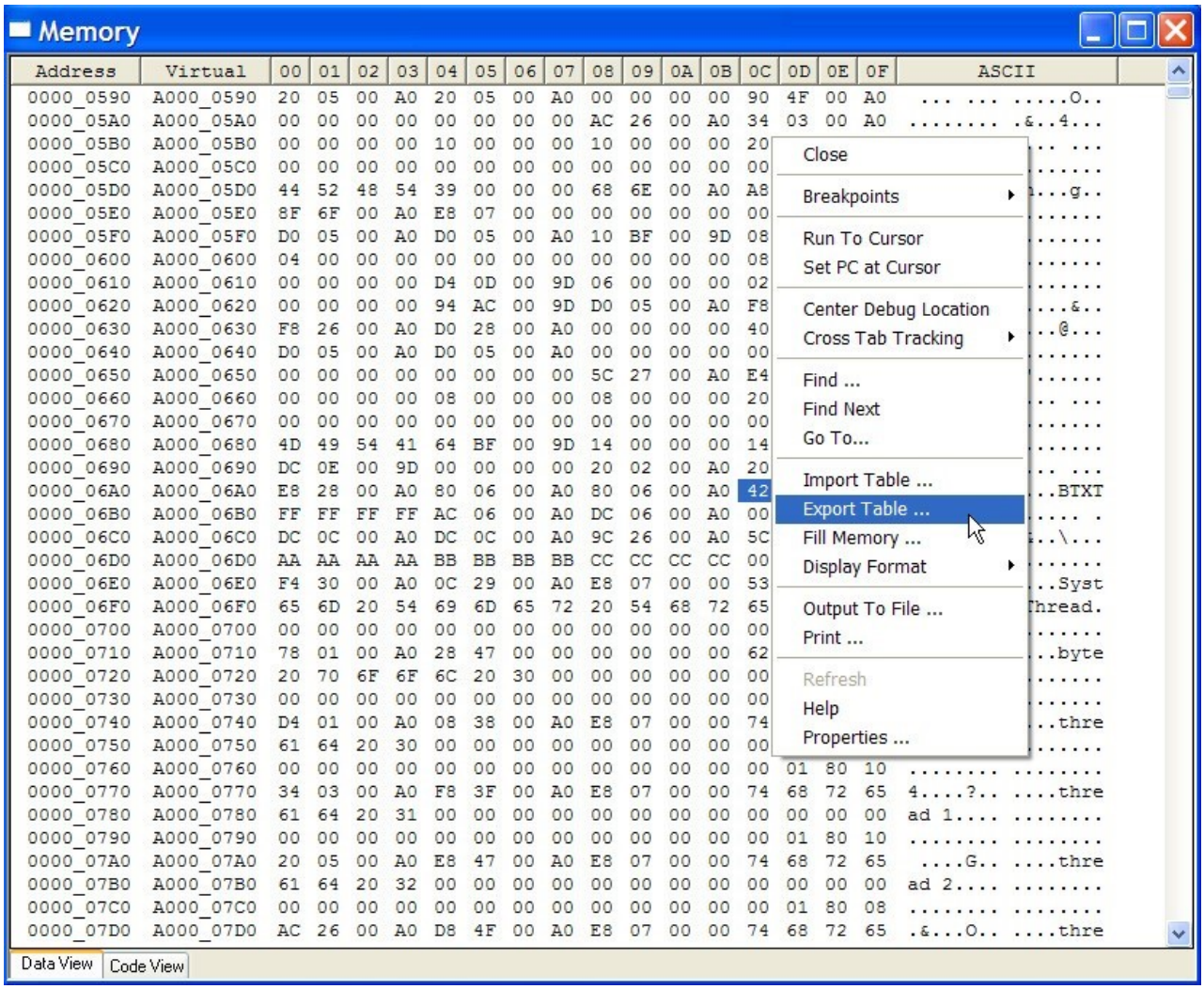




Step 4 This highlights the contents of the first location of the trace buffer, which is always the string BTXT....



Step 5 Now, right-click again to bring up the options menu, and select **Export Table**.



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).

![Screenshot of the Export As dialog.

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:releasethreadxdemo_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.

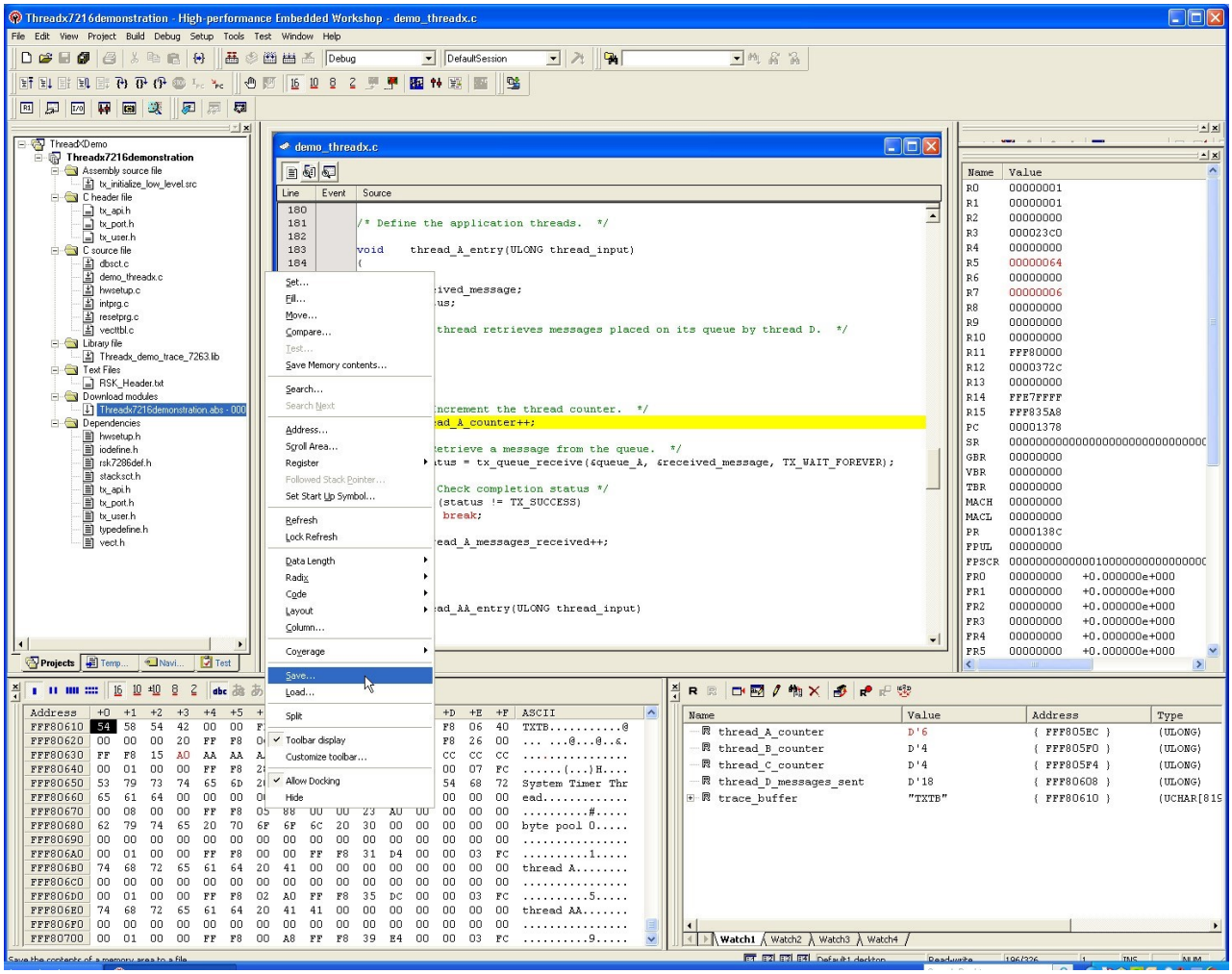
Renesas HEW

The trace buffer can be dumped to a host file easily with the Renasas HEW tools by following the three steps (and substeps) below:

Step 1 Open Memory Window.

![Screenshot of the 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