
Eclipse ThreadX RTOS

This advanced real-time operating system (RTOS) is designed specifically for deeply embedded applications. Among the multiple benefits it provides are advanced scheduling facilities, message passing, interrupt management, and messaging services. Eclipse ThreadX RTOS has many advanced features, including picokernel architecture, preemption threshold, event chaining, and a rich set of system services.

Here are the key features and modules of ThreadX:



Eclipse ThreadX RTOS

- **Small:** ~2KB Minimal Footprint
- **Fast:** Sub microsecond context switch, APIs
- **Safe:** SIL 4, ASIL D, Medical Class C
- **Advanced:** Preemption-threshold, Event Chaining, Auto Scaling, ThreadX Modules w/memory protection
- **Easy:** Consistent API, Extensive out-of-box examples



Getting Started

Eclipse ThreadX has been integrated to the semiconductor's SDKs and development environment. You can develop using the tools of choice from STMicroelectronics, NXP, Renesas and Microchip.

We also provide getting started guide and samples using development boards from semiconductors you can build and test with.

See Overview of Eclipse ThreadX RTOS for the high-level overview.

Repository Structure and Usage

Directory layout

1	.		
2	cmake	# CMakefile files for building the	
	project		
3	common	# Core ThreadX files	
4	common_modules	# Core ThreadX module files	
5	common_smp	# Core ThreadX SMP files	
6	docs	# Documentation supplements	
7	ports	# Architecture and compiler specific	
	files. See below for directory breakdown		
8	cortex_m7		
9	iar	# Example IAR compiler sample project	
10	example build	# IAR workspace and sample project files	
11	inc	# tx_port.h for this architecture	
12	src	# Source files for this architecture	
13	ac6	# Example ac6/Keil sample project	
14	gnu	# Example gnu sample project	
15	...		
16	...		
17	ports_modules	# Architecture and compiler specific	
	files for threadX modules		
18	ports_smp	# Architecture and compiler specific	
	files for threadX SMP		
19	samples	# demo_threadx.c	
20	utility	# Test cases and utilities	

Branches & Releases

The master branch has the most recent code with all new features and bug fixes. It does not represent the latest General Availability (GA) release of the library. Each official release (preview or GA) will be tagged to mark the commit and push it into the Github releases tab, e.g. **v6.2-rel**.

When you see xx-xx-xxxx, 6.x or x.x in function header, this means the file is not officially released yet. They will be updated in the next release. See example below.

```

1  /*
   *****
   */
2  /*
   */
3  /*  FUNCTION                                RELEASE
   */

```

```

4  /*
   */
5  /*  _tx_initialize_low_level                      Cortex-M23/GNU
   */
6  /*                                          6.x
   */
7  /*  AUTHOR                                          */
8  /*
   */
9  /*  Scott Larson, Microsoft Corporation
   */
10 /*
   */
11 /*  DESCRIPTION                                          */
12 /*
   */
13 /*  This function is responsible for any low-level processor
   */
14 /*  initialization, including setting up interrupt vectors, setting
   */
15 /*  up a periodic timer interrupt source, saving the system stack
   */
16 /*  pointer for use in ISR processing later, and finding the first
   */
17 /*  available RAM memory address for tx_application_define.
   */
18 /*
   */
19 /*  INPUT                                          */
20 /*
   */
21 /*  None                                          */
22 /*
   */
23 /*  OUTPUT                                          */
24 /*
   */
25 /*  None

```

```

26  /*
27  /*  CALLS
28  /*
29  /*  None
30  /*
31  /*  CALLED BY
32  /*
33  /*  _tx_initialize_kernel_enter      ThreadX entry function
34  /*
35  /*  RELEASE HISTORY
36  /*
37  /*  DATE          NAME          DESCRIPTION
38  /*
39  /*  09-30-2020    Scott Larson    Initial Version 6.1
40  /*  xx-xx-xxxx    Scott Larson    Include tx_user.h,
41  /*                                     resulting in version 6.x
42  /*
43  /*
44  /*  *****
45  /*

```

Supported Architecture Ports

ThreadX

1	arc_em	cortex_a12	cortex_m0	cortex_r4
2	arc_hs	cortex_a15	cortex_m23	cortex_r5
3	arm11	cortex_a17	cortex_m3	cortex_r7
4	arm9	cortex_a34	cortex_m33	
5	c667x	cortex_a35	cortex_m4	
6	linux	cortex_a5	cortex_m55	
7	risc-v32	cortex_a53	cortex_m7	
8	rxv1	cortex_a55	cortex_m85	
9	rxv2	cortex_a57		
10	rxv3	cortex_a5x		
11	win32	cortex_a65		
12	xtensa	cortex_a65ae		
13		cortex_a7		
14		cortex_a72		
15		cortex_a73		
16		cortex_a75		
17		cortex_a76		
18		cortex_a76ae		
19		cortex_a77		
20		cortex_a8		
21		cortex_a9		

ThreadX Modules

Eclipse ThreadX Modules component provides an infrastructure for applications to dynamically load modules that are built separately from the resident portion of the application.

```
1 cortex_a35
2 cortex_a35_smp
3 cortex_a7
4 cortex_m0+
5 cortex_m23
6 cortex_m3
7 cortex_m33
8 cortex_m4
9 cortex_m7
10 cortex_r4
11 rxv2
```

ThreadX SMP

Eclipse ThreadX SMP is a high-performance real-time SMP kernel designed specifically for embedded

applications.

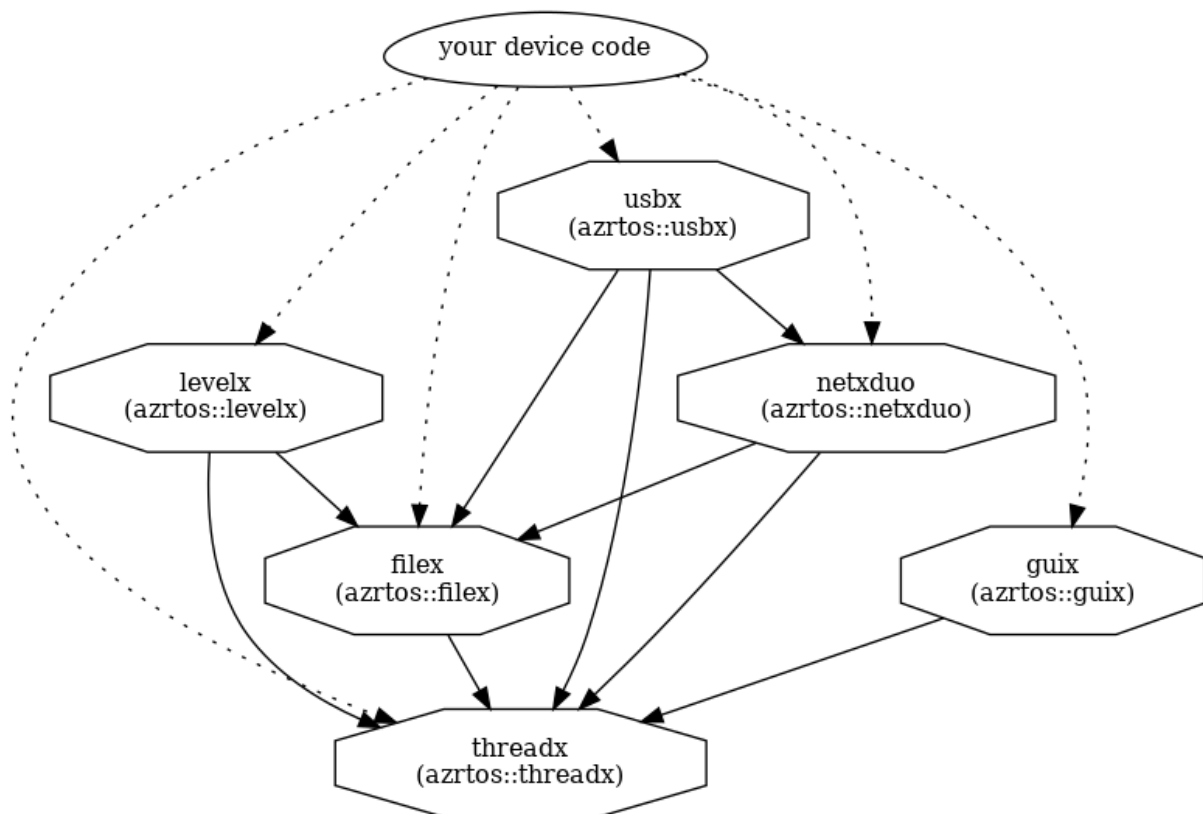
```
1  arc_hs_smp
2  cortex_a34_smp
3  cortex_a35_smp
4  cortex_a53_smp
5  cortex_a55_smp
6  cortex_a57_smp
7  cortex_a5x_smp
8  cortex_a5_smp
9  cortex_a65ae_smp
10 cortex_a65_smp
11 cortex_a72_smp
12 cortex_a73_smp
13 cortex_a75_smp
14 cortex_a76ae_smp
15 cortex_a76_smp
16 cortex_a77_smp
17 cortex_a78_smp
18 cortex_a7_smp
19 cortex_a9_smp
20 linux
```

Adaptation layer for ThreadX

ThreadX is an advanced real-time operating system (RTOS) designed specifically for deeply embedded applications. To help ease application migration to ThreadX RTOS, Eclipse ThreadX provides adaptation layers for various legacy RTOS APIs (FreeRTOS, POSIX, OSEK, etc.).

Component dependencies

The main components of ThreadX RTOS are each provided in their own repository, but there are dependencies between them, as shown in the following graph. This is important to understand when setting up your builds.



You will have to take the dependency graph above into account when building anything other than ThreadX itself.

Building and using the library

Instruction for building the ThreadX as static library using Arm GNU Toolchain and CMake. If you are using toolchain and IDE from semiconductor, you might follow its own instructions to use ThreadX RTOS components as explained in the Getting Started section.

1. Install the following tools:

- CMake version 3.0 or later
- Arm GNU Toolchain for arm-none-eabi
- Ninja

2. Cloning the repo

```
1 $ git clone https://github.com/eclipse-threadx/threadx.git
```

3. Define the features and addons you need in `tx_user.h` and build together with the component source code. You can refer to `tx_user_sample.h` as an example.

4. Building as a static library

Each component of ThreadX RTOS comes with a composable CMake-based build system that supports many different MCUs and host systems. Integrating any of these components into your device app code is as simple as adding a git submodule and then including it in your build using the CMake `add_subdirectory()`.

While the typical usage pattern is to include ThreadX into your device code source tree to be built & linked with your code, you can compile this project as a standalone static library to confirm your build is set up correctly.

An example of building the library for Cortex-M4:

```
1 $ cmake -Bbuild -GNinja -DCMAKE_TOOLCHAIN_FILE=cmake/cortex_m4.  
    cmake .  
2  
3 $ cmake --build ./build
```

Licensing

License terms for using Eclipse ThreadX are defined in the LICENSE.txt file of this repo. Please refer to this file for all definitive licensing information for all content, incl. the history of this repo.

Resources

The following are references to additional ThreadX RTOS resources:

- **Product introduction:** <https://github.com/eclipse-threadx/rtos-docs>
- **Product issues and bugs, or feature requests:** <https://github.com/eclipse-threadx/threadx/issues>
- **TraceX Installer:** <https://aka.ms/azrtos-tracex-installer>

You can also check previous questions or ask new ones on StackOverflow using the `threadx-rtos` and `threadx` tags.

Security

Eclipse ThreadX provides OEMs with components to secure communication and to create code and data isolation using underlying MCU/MPU hardware protection mechanisms. It is ultimately the responsibility of the device builder to ensure the device fully meets the evolving security requirements associated with its specific use case.

Contribution

Please follow the instructions provided in the CONTRIBUTING.md for the corresponding repository.