8/30/2020

NexOS Quick Start Guide

Table of Contents

1.0 Introduction....1
1.1 Release Notes....1
1.2 License....1
2.0 Required Software....2
2.1 Port Specific Software....2
2.1.1 PIC32MX Port....3
2.2 Required Files....3
2.3 Project Structure....4
2.4 Program Structure....6
2.5 Closing Notes....7

1.0 Introduction

The goal of this document is to provide the bare minimum amount of information to successfully get a project up and running with the NexOS. It does not go over customization, configuration, functionality, or documentation of the NexOS. It just lists the steps necessary to setup and compile a project with the NexOS. These instructions are valid for kernel version 1.00.00.

1.1 Release Notes

August 30, 2020 – Original release.

1.2 License

Below is the license associated with use the of NexOS.

2.0 Required Software

The latest NexOS project must be downloaded from GitHub at the following location https://GitHub.com/pic32/NexOS. This provides the source code for the RTOS along with example projects. Below is a breakdown of the different projects and folders of the repository downloaded from GitHub.

Example Projects:

- Simple Task Creation
- External Event
- Event Timer
- Callback Timer
- Handling Exceptions
- Simple Semaphore
- Pipe Communication
- Task Check In
- Advanced Task Creation

The NexOS folder contains the source code for the RTOS. The contents of this folder must be kept intact to be able to successfully compile the RTOS. The Generic Libraries folder contains code that the NexOS needs to successfully compile. This code is not specific to the OS and could be reused for other applications. The Generic Libraries folder should be located in the same folder as the NexOS.

The following section is the unique software required for each port of the NexOS.

2.1 Port Specific Software

Each port has individual dependencies based off of the microcontroller/CPU. The following sections outline each supported port of the NexOS along with the required software to compile the port.

2.1.1 PIC32MX Port

At the time of this document the following versions of software were used to verify the PIC32MX port.

- MPLAB X IDE v5.40 https://www.microchip.com/mplab/mplab-x-ide
- XC32 v2.41 https://www.microchip.com/mplab/compilers

The PIC32 Legacy Peripheral Libraries (located at the bottom of the compiler webpage) are also required to run the PIC32MX port. These libraries must be installed in the same directory as the compiler being used. If the XC32 compiler is located at F:\Microchip\xc32\v2.41, then the libraries should be installed in the location F:\Microchip\xc32\v2.41 too. There's been rumor in far and remote corners of the internet about PLIB warnings induced psychosis when people did not include the _SUPPRESS_PLIB_WARNING option in the build. If you want to avoid this condition please be sure to include it in the build options.

If any of the above software is outdated at the time of you reading this, then it can be found in the below location on Microchips software archive website page:

https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive

2.2 Required Files

The following list are the minimum required source files in order to use the NexOS.

- Kernel.c
- KernelTasks.c
- DoubleLinkedList.c
- Memory.c
- ContexSwitch.S
- CriticalSection.c
- Port.c
- Task.c

The following list is the minimum required header files in order to use the NexOS.

- Kernel.h
- KernelTasks.h
- DoubleLinkedList.h
- Memory.h
- CriticalSection.h
- Port.h
- Task.h
- TaskObject.h
- RTOSConfig.h
- DoubleLinkedListConfig.h

The file RTOSConfig.h is required in each project and should be added to the project (not referenced). This file is what allows the user to configure the RTOS as needed for the application. The RTOSConfig.h file found in the example program SimpleTaskCreation shows almost the bare minimum configuration to compile the NexOS (USING_MILLISECONDS_TO_TICKS_METHOD, USING_TASK_DELAY_TICKS_METHOD, and USING_KERNEL_VERSION_TO_STRING are not required to compile the NexOS). From here you can modify the configuration file as you need to.

2.3 Project Structure

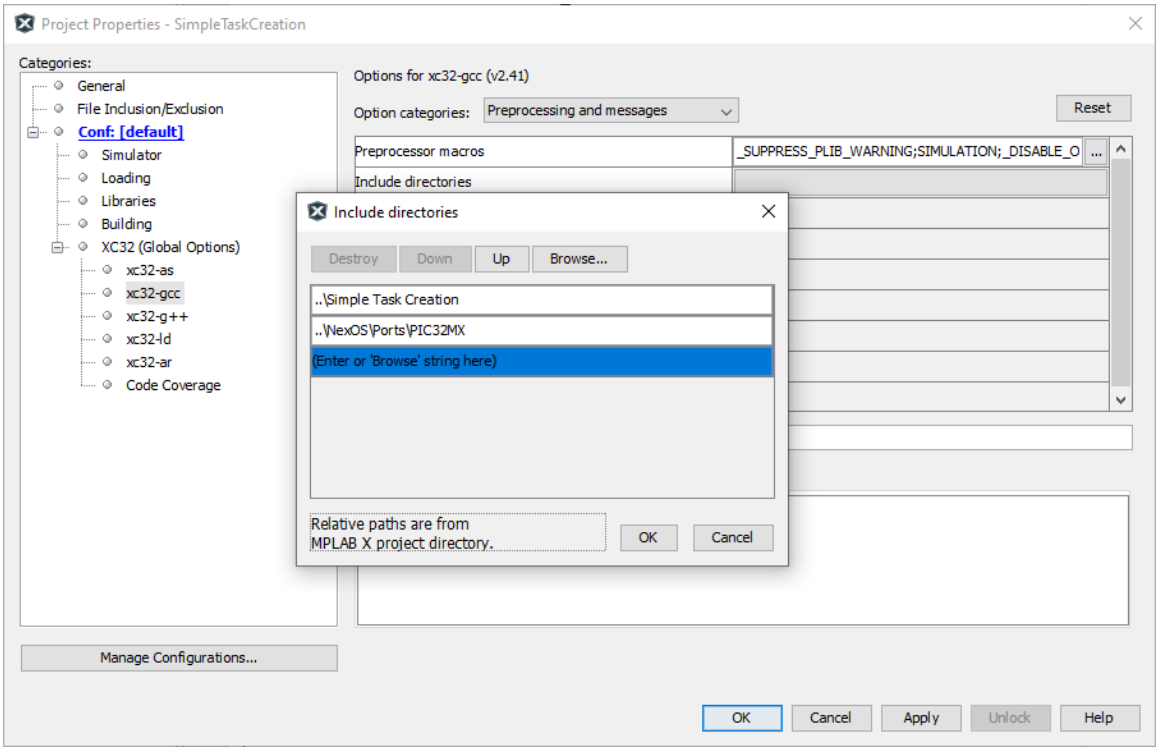
The directory structure used by the NexOS files requires them to stay together in the folder labeled NexOS. This is due to includes that utilize the .. in the path. Below is an example of this in CriticalSection.c

#include "../../Kernel/Kernel.h"

Due to this organization do not add the OS files to your project, only reference them. The below are the include directories required to build the projects:

- [PROJECT_FOLDER]
- NexOS\Ports\[PORT_NAME]

The PROJECT_FOLDER is so that the NexOS files can see the RTOSConfig.h the user has created. This path should be where the project resides. The NexOS\Ports\[PORT_NAME] is required because certain files in the NexOS reference the functions in the port files. PORT_NAME for the PIC32MX series should be PIC32MX (the name of the folder in Ports). Any file which is a configuration file (these files contain the word config in the name) should be located in the PROJECT_FOLDER.



Below is a picture of how these includes were done in MPLAB X IDE for the SimpleTaskCreation project.

The NexOS and Generic Libraries folders must also be contained in the same directory. Below is an example directory of multiple projects which reference the NexOS.

Controlled Projects
|
–> Advanced Task Creation
|
–> Handling Exceptions
|
–> NexOS
|
–> Generic Libraries

For example, given the above information and folder names the PROJECT_FOLDER for one of the projects is “Advanced Task Creation”. The only reason the NexOS folder and Generic Libraries folders are in the same location is because in the main.c file for the example projects reference the NexOS source files as #include “../NexOS/Kernel/Kernel.h”. If you wanted to have the NexOS folder and Generic Libraries folder (remember these two folders always have to stay together in the same directory) then that is fine. You will just need to add the paths the to the include directories list of your project and you won’t need to reference the source files as #include “../NexOS/Kernel/Kernel.h” but instead #include “Kernel.h”.

2.4 Program Structure

There are 2 essential OS methods that must be called to get the NexOS ready and running. These are the methods InitOS() and StartOSScheduler(). InitOS() must be called before any other method to the OS is called. Once the method StartOSScheduler() is called the NexOS will enable the system timer and look for the highest priority TASK to execute from. Below is a rough example of what the smallest program may look like.

```
#include "../NexOS/Kernel/Task.h"

void TaskCode(void *Args)
{
	while(1)
	{
		// do something
	}
}

int main(void)
{
	// initialize the OS
	InitOS();

	// create 1 TASK
	if(CreateTask(TaskCode, 300, 1, (void*)NULL, (TASK*)NULL) == (TASK*)NULL)
	{
		// handle the error
	}

	// now start the OS and scheduler
	StartOSScheduler();

	// you will never return to here after StartOSScheduler() is called
}
```

Technically no TASKs need to be made by the user and are totally optional. You could just use the Idle Task and/or EVENT callbacks along with CALLBACK_TIMERs and/or EVENT_TIMERs. This would make for a rather dull environment but it is still valid. In all practicality you will be creating at least 1 or more TASKs when using any RTOS.

2.5 Closing Notes

Below are some rules to make sure you follow for proper OS behavior.

- Never call a method that begins with OS.
- If any calls are made to the OS from an interrupt service routine be sure that the method name has ISR in it. Otherwise the OS method should not be called from an interrupt service routine.
- Never call a method that may block from the Idle Task. This is because the OS scheduler makes the assumption that at least 1 valid TASK is always eligible to run.
- Never call a method that may block from a user callback method.
- Only OS methods that have ISR in their name should be called from a user callback.
- Make sure any config file is located locally in your project. This allows multiple projects to use different configurations for the same libraries or OS.