Getting Started with the Novos RRS Environment

Processor Port: Cortex-M0 / M0+

Thank you for selecting the Novos RRS environment for the indicated processor port in your embedded system application. After uncompressing the Novos Delivery Package to your disk drive, the directory structure should look like the following diagram.

```
/.../NOVOS_RRS/
            |
            + /Base_Source/
            |
            + /Example Application/
            |
            + /Example Drivers/
            |
            + /Includes/
            |
            + /Tools/
                |
                + /ARM_Keil/
                |
                + /IAR_EWARM/
                |
                + /LPCXpresso/
```

Please note that the name of the root directory of the Novos Delivery Package may be different than that shown in the descriptions above and below. The directory name may contain the version number and license type of the Novos RRS environment. You may use the name as-is or you may also select a root directory name and path of your own choosing.

Three of the folders contain the code and header files you will need for compiling the Novos RRS code. They are:

- `Novos_RRS/Base_Source/`
- `Novos_RRS/Includes/`
- `Novos_RRS/Tools/`

The `/Base_Source/` folder consists of files that are mostly processor-independent. For the Novos RRS environment, they are:

- `APIs for Alarms and Counter Classes.c`
- `APIs for Event Group Class.c`
- `APIs for Foreground Entity Class.c`
- `APIs for Memory Pool Class.c`
- `APIs for Mutex Class.c`
- `APIs for Queue Class.c`
- `APIs for Semaphore Class.c`
- `APIs for Special Services Class.c`
- `APIs for Task Class.c`
- `Core_Functions.c`
- `PORT_Functions.c`
- `System_Tables.c`

The file, PORT_Functions.c, contains code for Novos RRS environment functions that is processor- and/or toolchain-specific. This file is mostly C code but contains some sections that are written in assembly language. All other files are C code.

The /Includes/ folder contains the header files that are required to compile a working Novos RRS environment:

- BasicNovosTypedefs.h
- BasicTypedefs.h
- Core_api.h
- ENUMcodes.h
- InternalTypedefs.h
- Novos_api.h
- NovosConfig.h
- NovosObjectProps.h
- NovosObjectStructs.h
- NSCCcodes.h
- NovosAppHdrs.h

The /Tools/ folder contains one or more sub-folders pertaining to different toolchains used for compiling the Novos RRS environment for the processor port. The following example shows an organization for the processor:

```
/Tools/
   |
   + /ARM Keil/
   |
   + /IAR EWARM/
   |
   + /NXP LPCXpresso/
   |
   ...
```

Each toolchain-specific sub-folder contains a special header file, TOOL_CPUspecs.h, which provides specifics about the tools and processor when compiling the Novos RRS environment. Choose the toolchain you are using and add the corresponding TOOL_CPUspecs.h file to the set of included header files from the /Includes/ folder when you compile the Novos RRS environment code.

Configuring the Novos RRS Environment

By design, the Novos RRS environment requires very little configuration in order to use it. All configuration elements are located in a single file, NovosConfig.h. There are definitions you can change and some that must remain unchanged. However, you may not need to make any changes as each of them has a default value. Specifically, the definitions you can change are:

Configuration Parameter	Default Assignment
DIAGNOSTICS	undefined
RETURN_FROM_TE_MANAGER	undefined
LEVELS_x	LEVELS_8
Structure for ENUMS of type DIH_PRIORITY	Three software priorities for Deferred Interrupt Handlers: HIGH_PRI, MEDIUM_PRI and LOW_PRI

Configuration Parameter	Default Assignment
Structure for ENUMS of type `FGT_PRIORITY`	Four software priorities for Foreground Tasks: `FGT_PRI_1`, `FGT_PRI_2`, `FGT_PRI_3` and `FGT_PRI_4`

You should open the `NovosConfig.h` file and read through it, paying attention to the comments associated with each parameter. If the default values are acceptable for your application, you are ready to move on to compiling the Novos RRS code.

If you need to change one of the parameters in the table above, make the change according to the directions in the comments of `NovosConfig.h`. Make no other changes to this file. When you have finished, you will be ready to compile the code.

Note that these parameters may have effects on both RAM requirements and performance. It is highly recommended that you fully understand any changes you make. Complete explanations of each configuration parameter are found in the Novos RRS Environment User Guide.

Compiling the Novos RRS Code

The Novos RRS code is written in such a way to support multiple processors and it makes use of the keywords `__near` and `__far`. For this processor port, the use of `__near` and `__far` are not required and will be rejected by the compilers, causing compilation errors. You should eliminate those keywords (symbols) before compiling the Novos RRS code. To eliminate them, you can edit the Novos RRS source code to remove all instances of the keywords (not recommended) or you can redefine them (recommended).

If you are using an Integrated Development Environment (IDE) such as those in the /tools/ folder, the Properties of the IDE Project provide a way to define a symbol. For the two symbols, `__near` and `__far`, redefine them by specifying them to be replaced by a `<space>` character.

- Define the symbol `__near` as a `<space>` character by the string "`__near=` "
- Define the symbol `__far` as a `<space>` character by the string "`__far=` "

Once you have made these two definitions in addition to the other Properties of the Project (or built a Make file), you should be able to compile the Novos RRS code and either archive it as a library or use it directly with your application code.

Application Code Preparation

Any application code module that contains references to Novos RRS services should include the following line of code to include the C header files needed to compile properly.

```
#include NovosAppHdrs.h
```

This header file provides an easy way to include the necessary Novos RRS environment header files you will need for compiling your application:

```
#include "NovosConfig.h"
#include "BasicTypedefs.h"
#include "BasicNovosTypedefs.h"
#include "ENUMcodes.h"
```

```
#include "NSCCcodes.h"
#include "NovosObjectProps.h"
#include "Novos_api.h"
#include “TOOL_CPUspecs.h”
```

In addition to including `NovosAppHdrs.h`, you should also include any processor-specific header files and your application-specific header files.

Application Code in a Novos RRS Environment

The `/Example Application/` folder provides a template for an application that uses the Novos RRS environment. It is not the only possible model for such an application. If you have not already done so, consult the eBook “*Operating System Concepts for Embedded Applications*”, which is available free of charge on the Embedded Environments Co. website, www.ee-novos.com. You may find other models in the eBook that also fit your application’s needs.

One of the critical items you must do is to define the elements of the System Properties structure (`SYSPROP`) in your application code. The `SYSPROP` structure is defined in the header file `NovosObjectProps.h`. You will need to have these properties defined in order to initialize the Novos RRS environment, which must be done prior to invoking any Novos RRS service. A complete description of these properties is found in the Novos RRS User Guide.

Drivers for the Novos RRS Environment

In the `/Example Drivers/` folder, you will find one or more drivers that you may find beneficial for use with the processor. Specifically, there is a `SysTick` driver that you can use to obtain a time base for your application.

Other drivers may be included but may or may not work with your chosen processor without some re-work or other modifications. Regardless, they may still serve as useful models for related or similar device drivers.

Special Information About This Port

The Novos RRS environment requires a system timebase in order to operate. Therefore, it is not “tickless”. You will need to provide some sort of timer for this functionality. The `SysTick` driver found in the `/Example Drivers/` folder of this Novos RRS distribution is sufficient to meet this need.

The Novos RRS environment is designed to run with interrupts enabled. In doing so, the design makes use of the `PendSV` and `SVCall` interrupts in the Cortex-M0. The Novos service `Port_InitNovos()` sets those priorities so that `PendSV` has the lowest possible priority (3) and `SVCall` has the highest priority (0).

CAUTION: If you choose to change these settings, please do so carefully as the change may cause undesirable results in your application.

The Novos initialization service, `PORT_InitNovos()` gets the MSP at the point of entry and uses it to set PSP as the stack pointer for the Idle Task.

All Background Tasks run in Thread Mode using PSP (`CONTROL = 0x00000002`)

The top of the System Stack is stored in the MSP so that there will be an automatic stack switch from PSP to MSP when an interrupt occurs whilst in a Background Task. Foreground operations use MSP exclusively. If an interrupt occurs during execution of a Foreground entity, there is no stack switch and MSP remains the active stack pointer.

A Note about the Novos Code:

One of the primary goals of the Novos code project has been to develop a body of C code that can be used with any number of C compilers on any microcontroller to which it is ported, whether its data bus is 8-, 16- or 32-bits wide. To reach that goal required the use of more casting than one might ordinarily use. We wanted to make sure that the code compiled without errors or warnings and the extra casting made that possible. So, while it may look a bit different than code you might produce, it is portable, it compiles and works, which is our goal.

Problems?

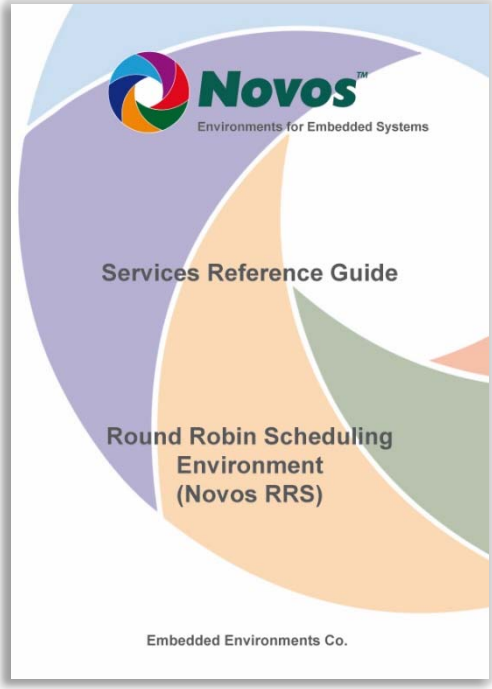
Even though we have taken pains to ensure the code performs as expected, it is still software and that means there could be flaws. If you do encounter a provable bug, please let us know about it. Your input helps improve the product not only for yourself but for other users as well. Please use the Novos Problem Report Form at…

https://www.ee-novos.com/problem-report-form/

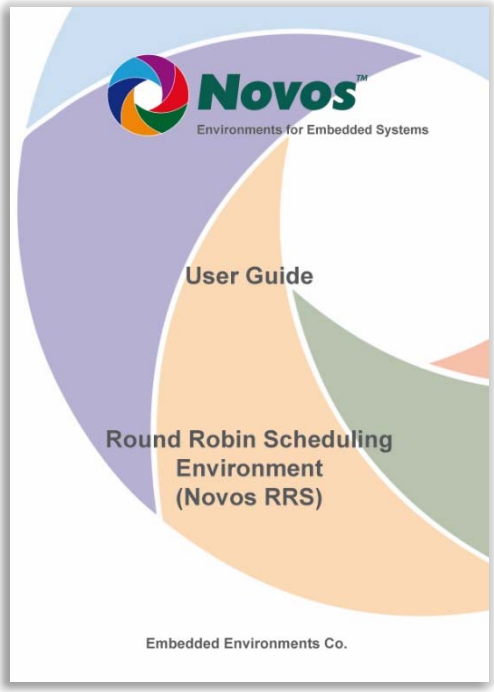
Be a Better Developer and Make a Better Application

Use of an operating system in an embedded application is best served by understanding how it works, its limitations and its services. Knowing what to do is just as important as knowing what not to do. Proper use of the operating system enhances the application development experience, improves the work product and saves time, all of which lead to a better resulting product. To aid you in efficient use of this Novos RRS environment, a Services Reference Guide and/or a User Guide are/is available for a nominal fee. Special pricing is offered when both manuals are purchased together. Sample versions of both are also available FREE of CHARGE. Whether you are ready to purchase or just want to take a high-level look at either document, click on the following link.

https://www.ee-novos.com/downloads/



The Services Reference Guide provides details about each Novos RRS service, including its prototype, an explanation of all calling and return parameters, options and a detailed description of its operation. You have the code to the Novos RRS environment but the Services Reference Guide will help you use it properly and efficiently. This document is available for immediate download to help you incorporate Novos within your embedded systems.



The Novos RRS User Guide provides a close look inside the Novos RRS environment, including its theory of operation, what you can and cannot do, a detailed description of its various object classes, a summary of supported services and much more. Use this document to guide you to good system design. It is available for immediate download.