23 Configuring a System

23.1 Introduction

RTEMS must be configured for an application. This configuration information encompasses a variety of information including the length of each clock tick, the maximum number of each RTEMS object that can be created, the application initialization tasks, the task scheduling algorithm to be used, and the device drivers in the application.

Although this information is contained in data structures that are used to RTEMS at system initialization time, the data structures themselves should only rarely to be generated by hand. RTEMS provides a macro based system which provides the standard and simple mechanism to automate the generation of these structures.

The RTEMS header file `<rtems/confdefs.h>` is the core of the automatic generation of system configuration. It is based on the idea of setting macros which define configuration parameters of interest to the application and defaulting or calculating all others. This variety of macros can automatically produce all of the configuration data required for an RTEMS application.

Trivia: `confdefs` is shorthand for a **Configuration Defaults**.

As a general rule, the application developer only specifies values for the configuration parameters of interest to them. They define what resources or features they require. In most cases, when a parameter is not specified, it defaults to zero (0) instances, a standards compliant value, or disabled as appropriate. For example, by default there will be 256 task priority levels but this can be lowered by the application. This number of priority levels is required to be standards compliant.

For each configuration parameter in the configuration tables, the macro corresponding to that field is discussed. It is expected that all systems can be easily configured using the `<rtems/confdefs.h>` mechanism. It is also expected that using this mechanism will avoid internal RTEMS configuration changes impacting applications.

23.2 Default Value Selection Philosophy

The user should be aware that the defaults are intentionally set as low as possible. By default, no application resources are configured. The `<rtems/confdefs.h>` file ensures that at least one application tasks or thread is configured and that at least one of the initialization task/thread tables is configured.

23.3 Sizing the RTEMS Workspace

The RTEMS Workspace is a user-specified block of memory reserved for use by RTEMS. The application should NOT modify this memory. This area consists primarily of the RTEMS data structures whose exact size depends upon the values specified in the Configuration Table. In addition, task stacks and floating point context areas are dynamically allocated from the RTEMS Workspace.

The `<rtems/confdefs.h>` mechanism calculates the size of the RTEMS Workspace automatically. It assumes that all tasks are floating point and that all will be allocated the

mininum stack space. This calculation also automatically includes the memory that will be allocated for internal use by RTEMS. In the event, there is an under-estimation of the amount of memoryy required, the `CONFIGURE_MEMORY_OVERHEAD` is provided as a work-around.

The starting address of the RTEMS Workspace is determined by the BSP must be aligned on at least a four-byte boundary. Failure to properly align the workspace area will result in the `rtems_fatal_error_occurred` directive being invoked with the `RTEMS_INVALID_ADDRESS` error code.

The file `<rtems/confdefs.h>` will calculate the value that is specified as the `work_space_size` parameter of the Configuration Table. There are many parameters the application developer can specify to help `<rtems/confdefs.h>` in its calculations. Correctly specifying the application requirements via parameters such as `CONFIGURE_EXTRA_TASK_STACKS` and `CONFIGURE_MAXIMUM_TASKS` is critical for production software.

The allocation of objects can operate in two modes. The default mode has an object number ceiling. No more than the specified number of objects can be allocated from the RTEMS Workspace. The number of objects specified in the particular API Configuration table fields are allocated at initialisation. The second mode allows the number of objects to grow to use the available free memory in the RTEMS Workspace.

See Section 23.5.1 [Configuring a System Unlimited Objects], page 254 for more details about the second mode, which allows for dynamic allocation of objects and therefore does not provide determinism. This mode is useful mostly for when the number of objects cannot be determined ahead of time or when porting software for which you do not know the object requirements.

Note that future versions of RTEMS may not have the same memory requirements per object. Although the value calculated is sufficient for a particular target processor and release of RTEMS, memory usage is subject to change across versions and target processors. To avoid problems, users should accurately specify each configuration parameter and allow `<rtems/confdefs.h>` to calculate the memory requirements. The memory requirements are likely to change each time one of the following events occurs:

- a configuration parameter is modified,
- task or interrupt stack requirements change,
- task floating point attribute is altered,
- RTEMS is upgraded, or
- the target processor is changed.

Failure to provide enough space in the RTEMS Workspace will result in the `rtems_fatal_error_occurred` directive being invoked with the appropriate error code.

23.4 Potential Issues with RTEMS Workspace Estimation

The `<rtems/confdefs.h>` file estimates the amount of memory required for the RTEMS Workspace. This estimate is only as accurate as the information given to `<rtems/confdefs.h>` and may be either too high or too low for a variety of reasons. Some of the reasons that `<rtems/confdefs.h>` may reserve too much memory for RTEMS are:

- All tasks/threads are assumed to be floating point.

Conversely, there are many more reasons, the resource estimate could be too low:

- Task/thread stacks greater than minimum size must be accounted for explicitly by developer.
- Memory for messages is not included.
- Device driver requirements are not included.
- Network stack requirements are not included.
- Requirements for add-on libraries are not included.

In general, `<rtems/confdefs.h>` is very accurate when given enough information. However, it is quite easy to use a library and forget to account for its resources.

23.5 Configuration Example

In the following example, the configuration information for a system with a single message queue, four (4) tasks, and a time slice fifty (50) milliseconds is as follows:

```
#include <bsp.h>

#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER

#define CONFIGURE_MICROSECONDS_PER_TICK   1000 /* 1 millisecond */
#define CONFIGURE_TICKS_PER_TIMESLICE       50 /* 50 milliseconds */

#define CONFIGURE_RTEMS_INIT_TASKS_TABLE

#define CONFIGURE_MAXIMUM_TASKS 4
#define CONFIGURE_MAXIMUM_MESSAGE_QUEUES 1

#define CONFIGURE_MESSAGE_BUFFER_MEMORY \
  CONFIGURE_MESSAGE_BUFFERS_FOR_QUEUE( 10, sizeof(struct USER_MESSAGE))

#define CONFIGURE_INIT
#include <rtems/confdefs.h>
```

In this example, only a few configuration parameters are specified. The impact of these are as follows:

- The example specified `CONFIGURE_RTEMS_INIT_TASK_TABLE` but did not specify any additional parameters. This results in a configuration which will begin execution at single initialization task named `Init` which is non-preemptible and at priority one (1).
- By specifying `CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER`, this application is configured to have a clock tick device driver. This is required for the passage of time including delays and wall time. Further configuration details about time are

provided. Per CONFIGURE_MICROSECONDS_PER_TICK and CONFIGURE_TICKS_PER_TIMESLICE, the user specified they wanted a clock tick to occur each millisecond, and that the length of a timeslice would be fifty (50) milliseconds.

- By specifying CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER, the application will include a console device driver. This provides for standard I/O) on at least /dev/console. Implicitly, the Configuration Defaults header file configures enough resources for three (3) file descriptors to be used for standard in, out, and error on a device that supports the POSIX *termios* interface.
- The example above specifies via CONFIGURE_MAXIMUM_TASKS, that the application requires a maximum of four (4) concurrently existent Classic API tasks. Similarly, by specifying CONFIGURE_MAXIMUM_MESSAGE_QUEUES, there may be a maximum of only one (1) concurrently existent Classic API message queues.
- The most surprising configuration parameter in this example is the use of CONFIGURE_MESSAGE_BUFFER_MEMORY. Message buffer memory is allocated from the RTEMS Workspace and must be accounted for. In this example, the single message queue will have up to twenty (20) messages of type struct USER_MESSAGE.
- The CONFIGURE_INIT constant must be defined in order to make <rtems/confdefs.h> instantiate the configuration data structures. This can only be defined in one source file per application that includes <rtems/confdefs.h> or the symbol table will be instantiated multiple times and linking errors produced.

This example illustrates that parameters have default values. Among other things, the application implicitly used the following defaults:

- All unspecified types of communications and synchronization objects in the Classic and POSIX Threads API have maximums of zero (0).
- The filesystem will be the default filesystem which only supports device nodes.
- The application will have the default number of priority levels.
- The minimum task stack size will be that recommended by RTEMS for the target architecture.

23.5.1 Unlimited Objects

In real-time embedded systems the RAM is normally a limited, critical resource and dynamic allocation is avoided as much as possible to ensure predictable, deterministic execution times. For such cases, see Section 23.3 [Configuring a System Sizing the RTEMS Workspace], page 251 for an overview of how to tune the size of the workspace. Frequently when users are porting software to RTEMS the precise resource requirements of the software is unknown. In these situations users do not need to control the size of the workspace very tightly because they just want to get the new software to run; later they can tune the workspace size as needed.

The following object classes in the Classic API can be configured in unlimited mode:

- Tasks
- Timers
- Semaphores

- Message Queues
- Periods
- Barriers
- Partitions
- Regions
- Ports

Additionally, the following object classes from the POSIX API can be configured in unlimited mode:

- Threads
- Mutexes
- Condition Variables
- Keys
- Timers
- Message Queues
- Message Queue Descriptors
- Semaphores
- Barriers
- Read/Write Locks
- Spinlocks

Due to how the POSIX object memory requirements are configured the unlimited object support does not provide unlimited size declarations for POSIX keys or queued signals.

Users are cautioned that using unlimited objects is not recommended for production software unless the dynamic growth is absolutely required. It is generally considered a safer embedded systems programming practice to know the system limits rather than experience an out of memory error at an arbitrary and largely unpredictable time in the field.

23.5.2 Per Object Class Unlimited Object Instances

When the number of objects is not known ahead of time, RTEMS provides an auto-extending mode that can be enabled individually for each object type by using the macro `rtems_resource_unlimited`. This takes a value as a parameter, and is used to set the object maximum number field in an API Configuration table. The value is an allocation unit size. When RTEMS is required to grow the object table it is grown by this size. The kernel will return the object memory back to the RTEMS Workspace when an object is destroyed. The kernel will only return an allocated block of objects to the RTEMS Workspace if at least half the allocation size of free objects remain allocated. RTEMS always keeps one allocation block of objects allocated. Here is an example of using `rtems_resource_unlimited`:

```
#define CONFIGURE_MAXIMUM_TASKS rtems_resource_unlimited(5)
```

Object maximum specifications can be evaluated with the `rtems_resource_is_unlimited` and `rtems_resource_maximum_per_allocation` macros.

23.5.3 Unlimited Object Instances

To ease the burden of developers who are porting new software RTEMS also provides the capability to make all object classes listed above operate in unlimited mode in a simple manner. The application developer is only responsible for enabling unlimited objects and specifying the allocation size.

23.5.4 Enable Unlimited Object Instances

CONSTANT: CONFIGURE_OBJECTS_UNLIMITED

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

CONFIGURE_OBJECTS_UNLIMITED enables rtems_resource_unlimited mode for Classic API and POSIX API objects that do not already have a specific maximum limit defined.

NOTES:

When using unlimited objects, it is common practice to also specify CONFIGURE_UNIFIED_WORK_AREAS so the system operates with a single pool of memory for both RTEMS and application memory allocations.

23.5.5 Specify Unlimited Objects Allocation Size

CONSTANT: CONFIGURE_OBJECTS_ALLOCATION_SIZE

DATA TYPE: integer

RANGE: undefined or positive

DEFAULT VALUE: If not defined and CONFIGURE_OBJECTS_UNLIMITED is defined, the default value is eight (8).

DESCRIPTION:

CONFIGURE_OBJECTS_ALLOCATION_SIZE provides an allocation size to use for rtems_resource_unlimited when using CONFIGURE_OBJECTS_UNLIMITED.

NOTES:

By allowing users to declare all resources as being unlimited the user can avoid identifying and limiting the resources used. CONFIGURE_OBJECTS_UNLIMITED does not support varying the allocation sizes for different objects; users who want that much control can define the rtems_resource_unlimited macros themselves.

```
#define CONFIGURE_OBJECTS_UNLIMITED
#define CONFIGURE_OBJECTS_ALLOCATION_SIZE 5
```

23.6 Classic API Configuration

This section defines the Classic API related system configuration parameters supported by `<rtems/confdefs.h>`.

23.6.1 Specify Maximum Classic API Tasks

CONSTANT: `CONFIGURE_MAXIMUM_TASKS`

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

`CONFIGURE_MAXIMUM_TASKS` is the maximum number of Classic API Tasks that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

This calculations for the required memory in the RTEMS Workspace for tasks assume that each task has a minimum stack size and has floating point support enabled. The configuration parameter `CONFIGURE_EXTRA_TASK_STACKS` is used to specify task stack requirements **ABOVE** the minimum size required. See Section 23.10.7 [Configuring a System Reserve Task/Thread Stack Memory Above Minimum], page 273 for more information about `CONFIGURE_EXTRA_TASK_STACKS`.

The maximumm number of POSIX threads is specified by `CONFIGURE_MAXIMUM_POSIX_THREADS`. See Section 23.8.1 [Configuring a System Specify Maximum POSIX API Threads], page 264 for more details.

A future enhancement to `<rtems/confdefs.h>` could be to eliminate the assumption that all tasks have floating point enabled. This would require the addition of a new configuration parameter to specify the number of tasks which enable floating point support.

23.6.2 Disable Classic API Notepads

CONSTANT: `CONFIGURE_DISABLE_CLASSIC_API_NOTEPADS`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not defined and Classic API Notepads are supported.

DESCRIPTION:

`CONFIGURE_DISABLE_CLASSIC_API_NOTEPADS` should be defined if the user does not want to have support for Classic API Notepads in their application.

NOTES:

Disabling Classic API Notepads saves the allocation of sixteen (16) thirty-two bit integers. This saves sixty-four bytes per task/thread plus the allocation overhead. Notepads are rarely used in applications and this can save significant memory in a low RAM system.

23.6.3 Specify Maximum Classic API Timers

CONSTANT: `CONFIGURE_MAXIMUM_TIMERS`

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

`CONFIGURE_MAXIMUM_TIMERS` is the maximum number of Classic API Timers that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.6.4 Specify Maximum Classic API Semaphores

CONSTANT: `CONFIGURE_MAXIMUM_SEMAPHORES`

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

`CONFIGURE_MAXIMUM_SEMAPHORES` is the maximum number of Classic API Semaphores that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.6.5 Specify Maximum Classic API Message Queues

CONSTANT: `CONFIGURE_MAXIMUM_MESSAGE_QUEUES`

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

`CONFIGURE_MAXIMUM_MESSAGE_QUEUES` is the maximum number of Classic API Message Queues that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.6.6 Specify Maximum Classic API Barriers

CONSTANT: `CONFIGURE_MAXIMUM_BARRIERS`

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

`CONFIGURE_MAXIMUM_BARRIERS` is the maximum number of Classic API Barriers that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.6.7 Specify Maximum Classic API Periods

CONSTANT: `CONFIGURE_MAXIMUM_PERIODS`

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

`CONFIGURE_MAXIMUM_PERIODS` is the maximum number of Classic API Periods that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.6.8 Specify Maximum Classic API Partitions

CONSTANT: `CONFIGURE_MAXIMUM_PARTITIONS`

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

`CONFIGURE_MAXIMUM_PARTITIONS` is the maximum number of Classic API Partitions that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.6.9 Specify Maximum Classic API Regions

CONSTANT: CONFIGURE_MAXIMUM_REGIONS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_REGIONS is the maximum number of Classic API Regions that can be concurrently active.

NOTES:

None.

23.6.10 Specify Maximum Classic API Ports

CONSTANT: CONFIGURE_MAXIMUM_PORTS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_PORTS is the maximum number of Classic API Ports that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.6.11 Specify Maximum Classic API User Extensions

CONSTANT: CONFIGURE_MAXIMUM_USER_EXTENSIONS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_USER_EXTENSIONS is the maximum number of Classic API User Extensions that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.7 Classic API Initialization Tasks Table Configuration

The <rtems/confdefs.h> configuration system can automatically generate an Initialization Tasks Table named Initialization_tasks with a single entry. The following parameters control the generation of that table.

23.7.1 Instantiate Classic API Initialization Task Table

CONSTANT: CONFIGURE_RTEMS_INIT_TASKS_TABLE

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

CONFIGURE_RTEMS_INIT_TASKS_TABLE is defined if the user wishes to use a Classic RTEMS API Initialization Task Table. The table built by <rtems/confdefs.h> specifies the parameters for a single task. This is sufficient for applications which initialization the system from a single task.

By default, this field is not defined as the user MUST select their own API for initialization tasks.

NOTES:

The application may choose to use the initialization tasks or threads table from another API.

A compile time error will be generated if the user does not configure any initialization tasks or threads.

23.7.2 Specifying Classic API Initialization Task Entry Point

CONSTANT: CONFIGURE_INIT_TASK_ENTRY_POINT

DATA TYPE: rtems_task_entry

RANGE: valid method pointer

DEFAULT VALUE: By default the value is Init.

DESCRIPTION:

CONFIGURE_INIT_TASK_ENTRY_POINT is the entry point (a.k.a. function name) of the single initialization task defined by the Classic API Initialization Tasks Table.

NOTES:

The user must implement the method Init or the method name provided in this configuration parameter.

23.7.3 Specifying Classic API Initialization Task Name

CONSTANT: CONFIGURE_INIT_TASK_NAME

DATA TYPE: rtems_name

RANGE: any value

DEFAULT VALUE: By default the value is `rtems_build_name( 'U', 'I', '1', ' ' )`.

DESCRIPTION:

CONFIGURE_INIT_TASK_NAME is the name of the single initialization task defined by the Classic API Initialization Tasks Table.

NOTES:

None.

23.7.4 Specifying Classic API Initialization Task Stack Size

CONSTANT: CONFIGURE_INIT_TASK_STACK_SIZE

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: By default value is the configured minimum stack size.

DESCRIPTION:

CONFIGURE_INIT_TASK_STACK_SIZE is the stack size of the single initialization task defined by the Classic API Initialization Tasks Table.

NOTES:

If the stack size specified is greater than the configured minimum, it must be accounted for in CONFIGURE_EXTRA_TASK_STACKS. See Section 23.10.7 [Configuring a System Reserve Task/Thread Stack Memory Above Minimum], page 273 for more information about CONFIGURE_EXTRA_TASK_STACKS.

23.7.5 Specifying Classic API Initialization Task Priority

CONSTANT: CONFIGURE_INIT_TASK_PRIORITY

DATA TYPE: rtems_task_priority

RANGE: 1 to CONFIGURE_MAXIMUM_PRIORITY

DEFAULT VALUE: By default the value is one (1) which is the highest priority in the Classic API.

DESCRIPTION:

CONFIGURE_INIT_TASK_PRIORITY is the initial priority of the single initialization task defined by the Classic API Initialization Tasks Table.

NOTES:

None.

23.7.6 Specifying Classic API Initialization Task Attributes

CONSTANT: CONFIGURE_INIT_TASK_ATTRIBUTES

DATA TYPE: rtems_attributes

RANGE: valid task attribute sets

DEFAULT VALUE: By default the tvalue is RTEMS_DEFAULT_ATTRIBUTES.

DESCRIPTION:

CONFIGURE_INIT_TASK_ATTRIBUTES is the task attributes of the single initialization task defined by the Classic API Initialization Tasks Table.

NOTES:

None.

23.7.7 Specifying Classic API Initialization Task Modes

CONSTANT: CONFIGURE_INIT_TASK_INITIAL_MODES

DATA TYPE: rtems_mode

RANGE: valid task mode sets

DEFAULT VALUE: By default the value is RTEMS_NO_PREEMPT.

DESCRIPTION:

CONFIGURE_INIT_TASK_INITIAL_MODES is the initial execution mode of the single initialization task defined by the Classic API Initialization Tasks Table.

NOTES:

None.

23.7.8 Specifying Classic API Initialization Task Arguments

CONSTANT: CONFIGURE_INIT_TASK_ARGUMENTS

DATA TYPE: rtems_task_argument

RANGE: valid rtems_task_argument values

DEFAULT VALUE: By default the value is 0.

DESCRIPTION:

CONFIGURE_INIT_TASK_ARGUMENTS is the task argument of the single initialization task defined by the Classic API Initialization Tasks Table.

NOTES:

None.

23.7.9 Not Using Generated Initialization Tasks Table

CONSTANT: CONFIGURE_HAS_OWN_INIT_TASK_TABLE

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

CONFIGURE_HAS_OWN_INIT_TASK_TABLE is defined if the user wishes to define their own Classic API Initialization Tasks Table. This table should be named Initialization_tasks.

NOTES:

This is a seldom used configuration parameter. The most likely use case is when an application desires to have more than one initialization task.

23.8 POSIX API Configuration

The parameters in this section are used to configure resources for the RTEMS POSIX API. They are only relevant if the POSIX API is enabled at configure time using the --enable-posix option.

23.8.1 Specify Maximum POSIX API Threads

CONSTANT: CONFIGURE_MAXIMUM_POSIX_THREADS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_THREADS is the maximum number of POSIX API Threads that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

This calculations for the required memory in the RTEMS Workspace for threads assume that each thread has a minimum stack size and has floating point support enabled. The configuration parameter CONFIGURE_EXTRA_TASK_STACKS is used to specify thread stack requirements **ABOVE** the minimum size required. See Section 23.10.7 [Configuring a System Reserve Task/Thread Stack Memory Above Minimum], page 273 for more information about CONFIGURE_EXTRA_TASK_STACKS.

The maximum number of Classic API Tasks is specified by CONFIGURE_MAXIMUM_TASKS. See Section 23.6.1 [Configuring a System Specify Maximum Classic API Tasks], page 257 for more details.

All POSIX threads have floating point enabled.

23.8.2 Specify Maximum POSIX API Mutexes

CONSTANT: CONFIGURE_MAXIMUM_POSIX_MUTEXES

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_MUTEXES is the maximum number of POSIX API Mutexes that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.8.3 Specify Maximum POSIX API Condition Variables

CONSTANT: CONFIGURE_MAXIMUM_POSIX_CONDITION_VARIABLES

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_CONDITION_VARIABLES is the maximum number of POSIX API Condition Variables that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.8.4 Specify Maximum POSIX API Keys

CONSTANT: CONFIGURE_MAXIMUM_POSIX_KEYS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_KEYS is the maximum number of POSIX API Keys that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.8.5 Specify Maximum POSIX API Timers

CONSTANT: CONFIGURE_MAXIMUM_POSIX_TIMERS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_TIMERS is the maximum number of POSIX API Timers that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.8.6 Specify Maximum POSIX API Queued Signals

CONSTANT: CONFIGURE_MAXIMUM_POSIX_QUEUED_SIGNALS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_QUEUED_SIGNALS is the maximum number of POSIX API Queued Signals that can be concurrently active.

NOTES:

None.

23.8.7 Specify Maximum POSIX API Message Queues

CONSTANT: CONFIGURE_MAXIMUM_POSIX_MESSAGE_QUEUES

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_MESSAGE_QUEUES is the maximum number of POSIX API Message Queues that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.8.8 Specify Maximum POSIX API Message Queue Descriptors

CONSTANT: CONFIGURE_MAXIMUM_POSIX_MESSAGE_QUEUE_DESCRIPTORS

DATA TYPE: integer

RANGE: greater than or equal to CONFIGURE_MAXIMUM_POSIX_MESSAGES_QUEUES

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_MESSAGE_QUEUE_DESCRIPTORS is the maximum number of POSIX API Message Queue Descriptors that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

CONFIGURE_MAXIMUM_POSIX_MESSAGE_QUEUE_DESCRIPTORS should be greater than or equal to CONFIGURE_MAXIMUM_POSIX_MESSAGE_QUEUES.

23.8.9 Specify Maximum POSIX API Semaphores

CONSTANT: CONFIGURE_MAXIMUM_POSIX_SEMAPHORES

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_SEMAPHORES is the maximum number of POSIX API Semaphores that can be concurrently active.

NOTES:

None.

23.8.10 Specify Maximum POSIX API Barriers

CONSTANT: CONFIGURE_MAXIMUM_POSIX_BARRIERS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_BARRIERS is the maximum number of POSIX API Barriers that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.8.11 Specify Maximum POSIX API Spinlocks

CONSTANT: CONFIGURE_MAXIMUM_POSIX_SPINLOCKS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_SPINLOCKS is the maximum number of POSIX API Spinlocks that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.8.12 Specify Maximum POSIX API Read/Write Locks

CONSTANT: CONFIGURE_MAXIMUM_POSIX_RWLOCKS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: The default for this field is 0.

DESCRIPTION:

CONFIGURE_MAXIMUM_POSIX_RWLOCKS is the maximum number of POSIX API Read/Write Locks that can be concurrently active.

NOTES:

This object class can be configured in unlimited allocation mode.

23.9 POSIX Initialization Threads Table Configuration

The <rtems/confdefs.h> configuration system can automatically generate a POSIX Initialization Threads Table named POSIX_Initialization_threads with a single entry. The following parameters control the generation of that table.

23.9.1 Instantiate POSIX API Initialization Thread Table

CONSTANT:

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this field is not defined as the user MUST select their own API for initialization tasks.

DESCRIPTION:

CONFIGURE_POSIX_INIT_THREAD_TABLE is defined if the user wishes to use a POSIX API Initialization Threads Table. The table built by <rtems/confdefs.h> specifies the parameters for a single thread. This is sufficient for applications which initialization the system from a single task.

By default, this field is not defined as the user MUST select their own API for initialization tasks.

NOTES:

The application may choose to use the initialization tasks or threads table from another API.

A compile time error will be generated if the user does not configure any initialization tasks or threads.

23.9.2 Specifying POSIX API Initialization Thread Entry Point

CONSTANT: CONFIGURE_POSIX_INIT_THREAD_ENTRY_POINT

DATA TYPE: void *(*entry_point)(void *)

RANGE: valid method pointer

DEFAULT VALUE: By default the value is POSIX_Init.

DESCRIPTION:

CONFIGURE_POSIX_INIT_THREAD_ENTRY_POINT is the entry point (a.k.a. function name) of the single initialization thread defined by the POSIX API Initialization Threads Table.

NOTES:

The user must implement the method POSIX_Init or the method name provided in this configuration parameter.

23.9.3 Specifying POSIX API Initialization Thread Stack Size

CONSTANT: CONFIGURE_POSIX_INIT_THREAD_STACK_SIZE

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: By default value is twice the configured minimum stack size.

DESCRIPTION:

CONFIGURE_POSIX_INIT_THREAD_STACK_SIZE is the stack size of the single initialization thread defined by the POSIX API Initialization Threads Table.

NOTES:

If the stack size specified is greater than the configured minimum, it must be accounted for in CONFIGURE_EXTRA_TASK_STACKS. See Section 23.10.7 [Configuring a System Reserve Task/Thread Stack Memory Above Minimum], page 273 for more information about CONFIGURE_EXTRA_TASK_STACKS.

23.9.4 Not Using Generated POSIX Initialization Threads Table

CONSTANT: CONFIGURE_POSIX_HAS_OWN_INIT_THREAD_TABLE

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

CONFIGURE_POSIX_HAS_OWN_INIT_THREAD_TABLE is defined if the user wishes to define their own POSIX API Initialization Threads Table. This table should be named POSIX_Initialization_threads.

NOTES:

This is a seldom used configuration parameter. The most likely use case is when an application desires to have more than one initialization task.

23.10 Basic System Information

This section defines the general system configuration parameters supported by <rtems/confdefs.h>.

23.10.1 Separate or Unified Work Areas

CONSTANT: CONFIGURE_UNIFIED_WORK_AREAS

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is undefined which specifies that the C Program Heap and the RTEMS Workspace will be separate.

DESCRIPTION:

When defined, the C Program Heap and the RTEMS Workspace will be one pool of memory.

When not defined, there will be separate memory pools for the RTEMS Workspace and C Program Heap.

NOTES:

Having separate pools does have some advantages in the event a task blows a stack or writes outside its memory area. However, in low memory systems the overhead of the two pools plus the potential for unused memory in either pool is very undesirable.

In high memory environments, this is desirable when you want to use the RTEMS "unlimited" objects option. You will be able to create objects until you run out of all available memory rather then just until you run out of RTEMS Workspace.

23.10.2 Length of Each Clock Tick

CONSTANT: `CONFIGURE_MICROSECONDS_PER_TICK`

DATA TYPE: integer

RANGE: non-zero positive values

DEFAULT VALUE: When not defined, the clock tick quantum is configured to be 10,000 microseconds which is ten (10) milliseconds.

DESCRIPTION:

This constant is used to specify the length of time between clock ticks.

When the clock tick quantum value is too low, the system will spend so much time processing clock ticks that it does not have processing time available to perform application work. In this case, the system will become unresponsive.

The lowest practical time quantum varies widely based upon the speed of the target hardware and the architectural overhead associated with interrupts. In general terms, you do not want to configure it lower than is needed for the application.

The clock tick quantum should be selected such that it all blocking and delay times in the application are evenly divisible by it. Otherwise, rounding errors will be introduced which may negatively impact the application.

NOTES:

This configuration parameter has no impact if the Clock Tick Device driver is not configured.

There may be BSP specific limits on the resolution or maximum value of a clock tick quantum.

23.10.3 Specifying Timeslicing Quantum

CONSTANT: `CONFIGURE_TICKS_PER_TIMESLICE`

DATA TYPE: integer

RANGE: non-zero positive values

DEFAULT VALUE: If unspecified, this parameter defaults to fifty (50).

DESCRIPTION:

This configuration parameter specifies the length of the timeslice quantum in ticks for each task.

NOTES:

This configuration parameter has no impact if the Clock Tick Device driver is not configured.

23.10.4 Specifying the Number of Thread Priority Levels

CONSTANT: `CONFIGURE_MAXIMUM_PRIORITY`

DATA TYPE: integer

RANGE: Valid values for this configuration parameter must be one (1) less than than a power of two (2) between 4 and 256 inclusively. In other words, valid values are 3, 7, 31, 63, 127, and 255.

DEFAULT VALUE: By default, RTEMS must support 256 priority levels to be compliant with various standards. These priorities range from zero (0) to 255. Thus, the default value for this field is 255.

DESCRIPTION:

This configuration parameter specified the maximum numeric priority of any task in the system and one less that the number of priority levels in the system.

Reducing the number of priorities in the system reduces the amount of memory allocated from the RTEMS Workspace.

NOTES:

The numerically greatest priority is the logically lowest priority in the system and will thus be used by the IDLE task.

Priority zero (0) is reserved for internal use by RTEMS and is not available to applications.

With some schedulers, reducing the number of priorities can reduce the amount of memory used by the scheduler. For example, the Deterministic Priority Scheduler (DPS) used by default uses three pointers of storage per priority level. Reducing the number of priorities from 256 levels to sixteen (16) can reduce memory usage by about three (3) kilobytes.

23.10.5 Specifying the Minimum Task Size

CONSTANT: `CONFIGURE_MINIMUM_TASK_STACK_SIZE`

DATA TYPE: integer

RANGE: non-zero positive integer

DEFAULT VALUE: When not defined by the application, this is set to the recommended minimum stack size for this processor.

DESCRIPTION:

The configuration parameter is set to the number of bytes the application wants the minimum stack size to be for every task or thread in the system.

Adjusting this parameter should be done with caution. Examining the actual usage using the Stack Checker Usage Reporting facility is recommended.

NOTES:

This parameter can be used to lower the minimum from that recommended. This can be used in low memory systems to reduce memory consumption for stacks. However, this must be done with caution as it could increase the possibility of a blown task stack.

This parameter can be used to increase the minimum from that recommended. This can be used in higher memory systems to reduce the risk of stack overflow without performing analysis on actual consumption.

23.10.6 Configuring the Size of the Interrupt Stack

CONSTANT: CONFIGURE_INTERRUPT_STACK_SIZE

DATA TYPE: integer

RANGE: non-zero positive integer

DEFAULT VALUE: If not specified, the interrupt stack will be of minimum size. The default value is the configured minimum task stack size.

DESCRIPTION:

CONFIGURE_INTERRUPT_STACK_SIZE is set to the size of the interrupt stack. The interrupt stack size is often set by the BSP but since this memory may be allocated from the RTEMS Workspace, it must be accounted for.

NOTES:

In some BSPs, changing this constant does NOT change the size of the interrupt stack, only the amount of memory reserved for it.

Patches which result in this constant only being used in memory calculations when the interrupt stack is intended to be allocated from the RTEMS Workspace would be welcomed by the RTEMS Project.

23.10.7 Reserve Task/Thread Stack Memory Above Minimum

CONSTANT: CONFIGURE_EXTRA_TASK_STACKS

DATA TYPE: integer

RANGE: Undefined or positive

DEFAULT VALUE: When this is not defined, the default value is 0.

DESCRIPTION:

This configuration parameter is set to the number of bytes the applications wishes to add to the task stack requirements calculated by <rtems/confdefs.h>.

NOTES:

This parameter is very important. If the application creates tasks with stacks larger then the minimum, then that memory is NOT accounted for by <rtems/confdefs.h>.

23.10.8 Automatically Zeroing the RTEMS Workspace and C Program Heap

CONSTANT: CONFIGURE_ZERO_WORKSPACE_AUTOMATICALLY

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: Unless overridden by the BSP, this is not defined by default. The default is **NOT** to zero out the RTEMS Workspace or C Program Heap.

DESCRIPTION:

This macro indicates whether RTEMS should zero the RTEMS Workspace and C Program Heap as part of its initialization. If defined, the memory regions are zeroed. Otherwise, they are not.

NOTES:

Zeroing memory can add significantly to system boot time. It is not necessary for RTEMS but is often assumed by support libraries.

23.10.9 Enable The Task Stack Usage Checker

CONSTANT: `CONFIGURED_STACK_CHECKER_ENABLED`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not defined and thus stack checking is disabled.

DESCRIPTION:

This configuration parameter is defined when the application wishes to enable run-time stack bounds checking.

NOTES:

In 4.9 and older, this configuration parameter was named `STACK_CHECKER_ON`.

This increases the time required to create tasks as well as adding overhead to each context switch.

23.10.10 Specify Application Specific User Extensions

CONSTANT: `CONFIGURE_INITIAL_EXTENSIONS`

DATA TYPE: List of `rtems_extensions_table` entries

RANGE: Undefined or a list of one or more user extensions.

DEFAULT VALUE: This value is not defined by default.

DESCRIPTION:

If `CONFIGURE_INITIAL_EXTENSIONS` is defined by the application, then this application specific set of initial extensions will be placed in the initial extension table.

NOTES:

None.

23.11 Configuring Custom Task Stack Allocation

RTEMS allows the application or BSP to define its own allocation and deallocation methods for task stacks. This can be used to place task stacks in special areas of memory or to utilize a Memory Management Unit so that stack overflows are detected in hardware.

23.11.1 Custom Task Stack Allocator Initialization

CONSTANT: CONFIGURE_TASK_STACK_ALLOCATOR_INIT

DATA TYPE: method pointer

RANGE: NULL or valid pointer to a method

DEFAULT VALUE: The default value for this field is NULL which indicates that task stacks will be allocated from the RTEMS Workspace.

DESCRIPTION:

CONFIGURE_TASK_STACK_ALLOCATOR_INIT configures the initialization method for an application or BSP specific task stack allocation implementation.

NOTES:

A correctly configured system must configure the following to be consistent:

- CONFIGURE_TASK_STACK_ALLOCATOR_INIT
- CONFIGURE_TASK_STACK_ALLOCATOR
- CONFIGURE_TASK_STACK_DEALLOCATOR

23.11.2 Custom Task Stack Allocator

CONSTANT: CONFIGURE_TASK_STACK_ALLOCATOR

DATA TYPE: method pointer

RANGE: NULL or valid method pointer

DEFAULT VALUE: The default value for this field is NULL which indicates that task stacks will be allocated from the RTEMS Workspace.

DESCRIPTION:

CONFIGURE_TASK_STACK_ALLOCATOR may point to a user provided routine to allocate task stacks.

NOTES:

A correctly configured system must configure the following to be consistent:

- CONFIGURE_TASK_STACK_ALLOCATOR_INIT
- CONFIGURE_TASK_STACK_ALLOCATOR
- CONFIGURE_TASK_STACK_DEALLOCATOR

23.11.3 Custom Task Stack Deallocator

CONSTANT: CONFIGURE_TASK_STACK_DEALLOCATOR

DATA TYPE: method pointer

RANGE: undefined or valid pointer

DEFAULT VALUE: The default value for this field is NULL which indicates that task stacks will be allocated from the RTEMS Workspace.

DESCRIPTION:

CONFIGURE_TASK_STACK_DEALLOCATOR may point to a user provided routine to free task stacks.

NOTES:

A correctly configured system must configure the following to be consistent:

- CONFIGURE_TASK_STACK_ALLOCATOR_INIT
- CONFIGURE_TASK_STACK_ALLOCATOR
- CONFIGURE_TASK_STACK_DEALLOCATOR

23.12 Configuring Memory for Classic API Message Buffers

This section describes the configuration parameters related to specifying the amount of memory reserved for Classic API Message Buffers.

23.12.1 Calculate Memory for a Single Classic Message API Message Queue

CONSTANT: CONFIGURE_MESSAGE_BUFFERS_FOR_QUEUE

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: This macro is only used as input to

DESCRIPTION:

This is a helper macro which is used to assist in computing the total amount of memory required for message buffers. Each message queue will have its own configuration with maximum message size and maximum number of pending messages.

The interface for this macro is as follows:

```
CONFIGURE_MESSAGE_BUFFERS_FOR_QUEUE(max_messages, size_per)
```

Where `max_messages` is the maximum number of pending messages and `size_per` is the size in bytes of the user message.

NOTES:

This macro is only used in support of CONFIGURE_MESSAGE_BUFFER_MEMORY.

23.12.2 Reserve Memory for All Classic Message API Message Queues

CONSTANT: CONFIGURE_MESSAGE_BUFFER_MEMORY

DATA TYPE: integer summation macro

RANGE: undefined (zero) or calculation resulting in a positive integer

DEFAULT VALUE: By default, this is not defined and zero (0) memory is reserved.

DESCRIPTION:

This macro is set to the number of bytes the application requires to be reserved for pending Classic API Message Queue buffers.

NOTES:

The following illustrates how the help macro `CONFIGURE_MESSAGE_BUFFERS_FOR_QUEUE` can be used to assist in calculating the message buffer memory required. In this example, there are two message queues used in this application. The first message queue has maximum of 24 pending messages with the message structure defined by the type `one_message_type`. The other message queue has maximum of 500 pending messages with the message structure defined by the type `other_message_type`.

```
#define CONFIGURE_MESSAGE_BUFFER_MEMORY \
 (CONFIGURE_MESSAGE_BUFFERS_FOR_QUEUE( \
    24, sizeof(one_message_type) + \
  CONFIGURE_MESSAGE_BUFFERS_FOR_QUEUE( \
    500, sizeof(other_message_type) \
 )
```

23.13 Seldom Used Configuration Parameters

This section describes configuration parameters supported by `<rtems/confdefs.h>` which are seldom used by applications. These parameters tend to be oriented to debugging system configurations and providing work-arounds when the memory estimated by `<rtems/confdefs.h>` is incorrect.

23.13.1 Specify Memory Overhead

CONSTANT: `CONFIGURE_MEMORY_OVERHEAD`

DATA TYPE: undefined or integer

RANGE: zero or positive

DEFAULT VALUE: The default value is 0.

DESCRIPTION:

Thie parameter is set to the number of kilobytes the application wishes to add to the requirements calculated by `<rtems/confdefs.h>`.

NOTES:

This configuration parameter should only be used when it is suspected that a bug in `<rtems/confdefs.h>` has resulted in an underestimation. Typically the memory allocation will be too low when an application does not account for all message queue buffers or task stacks.

23.13.2 Do Not Generate Configuration Information

CONSTANT: `CONFIGURE_HAS_OWN_CONFIGURATION_TABLE`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

This configuration parameter should only be defined if the application is providing their own complete set of configuration tables.

NOTES:

None.

23.13.3 Specify Location of RTEMS Workspace

CONSTANT: `CONFIGURE_EXECUTIVE_RAM_WORK_AREA`

DATA TYPE: pointer

RANGE: NULL or valid pointer

DEFAULT VALUE: By default, this value is not defined indicating that the BSP is to determine the location of the RTEMS Workspace.

DESCRIPTION:

This configuration parameter is the base address of the RTEMS Workspace.

NOTES:

The BSP is responsible for setting this address. It is highly unlikely that an application could do this portably and reliably.

23.14 C Library Support Configuration

This section defines the file system and IO library related configuration parameters supported by `<rtems/confdefs.h>`.

23.14.1 Enable Malloc Family Statistics

CONSTANT: `CONFIGURE_MALLOC_STATISTICS`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is undefined by default and Malloc Statistics are disabled.

DESCRIPTION:

This configuration parameter is defined when the application wishes to enable the gathering of more detailed statistics on the C Malloc Family of routines.

NOTES:

None.

23.14.2 Specify Maximum Number of File Descriptors

CONSTANT: `CONFIGURE_LIBIO_MAXIMUM_FILE_DESCRIPTORS`

DATA TYPE: integer

RANGE: Zero or positive

DEFAULT VALUE: If not defined, the default value is either zero (0) or three if `CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER` is defined. Three file descriptors allows RTEMS to support standard input, output, and error I/O streams on `/dev/console`.

DESCRIPTION:

This configuration parameter is set to the maximum number of files that can be concurrently open.

NOTES:

In addition to the actual file descriptor data structures, the RTEMS Libio Support library requires a Classic API semaphore for each file descriptor as well as one to manage the set. Thus this configuration parameter implicitly impacts the configured number of Classic API semaphores configured for the application.

23.14.3 Disable POSIX Termios Support

CONSTANT: `CONFIGURE_TERMIOS_DISABLED`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not defined and resources are reserved for the termios functionality.

DESCRIPTION:

This configuration parameter is defined if the software implementing POSIX termios functionality is not going to be used by this application.

NOTES:

The termios support library should not be included in an application executable unless it is directly referenced by the application or a device driver.

23.14.4 Specify Maximum Termios Ports

CONSTANT: `CONFIGURE_NUMBER_OF_TERMIOS_PORTS`

DATA TYPE: integer

RANGE: zero or positive integer

DEFAULT VALUE: By default, this is set to 1 so a console port can be used.

DESCRIPTION:

This configuration parameter is set to the number of ports using the termios functionality. Each concurrently active termios port requires resources.

NOTES:

If the application will be using serial ports including, but not limited to, the Console Device (e.g. `CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER`), then it is highly likely that this configuration parameter should NOT be is defined.

23.15 File System Configuration Parameters

This section defines File System related configuration parameters.

23.15.1 Providing Application Specific Mount Table

CONSTANT: `CONFIGURE_HAS_OWN_MOUNT_TABLE`

DATA TYPE: Undefined or an array of type `rtems_filesystem_mount_table_t`.

RANGE: Undefined or an array of type `rtems_filesystem_mount_table_t`.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

This configuration parameter is defined when the application provides their own filesystem mount table. The mount table is an array of `rtems_filesystem_mount_table_t` entries pointed to by the global variable `rtems_filesystem_mount_table`. The number of entries in this table is in an integer variable named `rtems_filesystem_mount_table_t`.

NOTES:

None.

23.15.2 Configure miniIMFS as Root File System

CONSTANT: `CONFIGURE_USE_MINIIMFS_AS_BASE_FILESYSTEM`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE:

This value is not defined by default. If no other root file system configuration parameters are specified, the IMFS will be used as the root file system.

DESCRIPTION:

This configuration parameter is defined if the application wishes to use the reduced functionality miniIMFS as the root filesystem. This reduced version of the full IMFS does not include the capability to mount other file system types, but it does support directories, device nodes, and symbolic links.

NOTES:

The miniIMFS nodes and is smaller in executable code size than the full IMFS.

23.15.3 Configure devFS as Root File System

CONSTANT: `CONFIGURE_USE_DEVFS_AS_BASE_FILESYSTEM`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This value is not defined by default. If no other root file system configuration parameters are specified, the IMFS will be used as the root file system.

DESCRIPTION:

This configuration parameter is defined if the application wishes to use the device-only filesytem as the root file system.

NOTES:

The device-only filesystem supports only device nodes and is smaller in executable code size than the full IMFS and miniIMFS.

The devFS is comparable in functionality to the pseudo-filesystem name space provided before RTEMS release 4.5.0.

23.15.4 Disable File System Support

CONSTANT:

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This value is not defined by default. If no other root file system configuration parameters are specified, the IMFS will be used as the root file system.

DESCRIPTION:

`CONFIGURE_APPLICATION_DISABLE_FILESYSTEM`

NOTES:

This configuration parameter is defined if the application dose not intend to use any kind of filesystem support. This include the device infrastructure necessary to support `printf()`.

23.16 BSP Specific Settings

This section describes BSP specific configuration settings used by `<rtems/confdefs.h>`. The BSP specific configuration settings are defined in `<bsp.h>`.

23.16.1 Disable BSP Configuration Settings

CONSTANT: `CONFIGURE_DISABLE_BSP_SETTINGS`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

All BSP specific configuration settings can be disabled by the application with the `CONFIGURE_DISABLE_BSP_SETTINGS` option.

NOTES:

None.

23.16.2 Specify BSP Supports sbrk()

CONSTANT: `CONFIGURE_MALLOC_BSP_SUPPORTS_SBRK`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This configuration is undefined by default.

DESCRIPTION:

This configuration parameter is defined by a BSP to indicate that it does not allocate all available memory to the C Program Heap used by the Malloc Family of routines.

If defined, when `malloc()` is unable to allocate memory, it will call the BSP supplied `sbrk()` to obtain more memory.

NOTES:

This parameter should not be defined by the application. Only the BSP knows how it allocates memory to the C Program Heap.

23.16.3 Specify BSP Specific Idle Task

CONSTANT: `BSP_IDLE_TASK_BODY`

DATA TYPE: Pointer to method.

RANGE: Null or pointer to method.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

If `BSP_IDLE_TASK_BODY` is defined by the BSP and `CONFIGURE_IDLE_TASK_BODY` is not defined by the application, then this BSP specific idle task body will be used.

NOTES:

As it has knowledge of the specific CPU model, system controller logic, and peripheral buses, a BSP specific IDLE task may be capable of turning components off to save power during extended periods of no task activity

23.16.4 Specify BSP Suggested Value for IDLE Task Stack Size

CONSTANT: `BSP_IDLE_TASK_STACK_SIZE`

DATA TYPE: integer

RANGE: undefined or positive integer

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

If `BSP_IDLE_TASK_STACK_SIZE` is defined by the BSP and `CONFIGURE_IDLE_TASK_STACK_SIZE` is not defined by the application, then this BSP suggested idle task stack size will be used.

NOTES:

The order of precedence for configuring the IDLE task stack size is:

- RTEMS default minimum stack size.
- If defined, then `CONFIGURE_MINIMUM_TASK_STACK_SIZE`.
- If defined, then the BSP specific `BSP_IDLE_TASK_SIZE`.
- If defined, then the application specified `CONFIGURE_IDLE_TASK_SIZE`.

23.16.5 Specify BSP Specific User Extensions

CONSTANT: `BSP_INITIAL_EXTENSION`

DATA TYPE: List of `rtems_extensions_table` entries

RANGE: Undefined or a list of one or more user extensions.

DEFAULT VALUE: This value is not defined by default.

DESCRIPTION:

If `BSP_INITIAL_EXTENSION` is defined by the BSP, then this BSP specific initial extension will be placed as the last entry in the initial extension table.

NOTES:

None.

23.16.6 Specifying BSP Specific Interrupt Stack Size

CONSTANT: `BSP_INTERRUPT_STACK_SIZE`

DATA TYPE:

RANGE:

DEFAULT VALUE:

DESCRIPTION:

If BSP_INTERRUPT_STACK_SIZE is defined by the BSP and CONFIGURE_INTERRUPT_STACK_SIZE is not defined by the application, then this BSP specific interrupt stack size will be used.

NOTES:

None.

23.16.7 Specifying BSP Specific Maximum Devices

CONSTANT: BSP_MAXIMUM_DEVICES

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: By default, this is not defined.

DESCRIPTION:

If BSP_MAXIMUM_DEVICES is defined by the BSP and CONFIGURE_MAXIMUM_DEVICES is not defined by the application, then this BSP specific maximum device count will be used. This option is specific to the device file system (devFS) and should not be confused with the CONFIGURE_MAXIMUM_DRIVERS option.

NOTES:

This parameter only impacts the devFS and thus is only used by <rtems/confdefs.h> when CONFIGURE_USE_DEVFS_AS_BASE_FILESYSTEM is specified.

23.16.8 BSP Recommends RTEMS Workspace be Cleared

CONSTANT: BSP_ZERO_WORKSPACE_AUTOMATICALLY

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

If BSP_ZERO_WORKSPACE_AUTOMATICALLY is defined by the BSP and CONFIGURE_ZERO_WORKSPACE_AUTOMATICALLY is not defined by the application, then the workspace will be zeroed automatically.

NOTES:

Zeroing memory can add significantly to system boot time. It is not necessary for RTEMS but is often assumed by support libraries.

23.16.9 Specify BSP Prerequisite Drivers

CONSTANT: CONFIGURE_BSP_PREREQUISITE_DRIVERS

DATA TYPE: array of device drivers

RANGE: Undefined or array of device drivers

DEFAULT VALUE: By default, this is not defined.

DESCRIPTION:

CONFIGURE_BSP_PREREQUISITE_DRIVERS is defined if the BSP has device drivers it needs to include in the Device Driver Table. This should be defined to the set of device driver entries that will be placed in the table at the **FRONT** of the Device Driver Table and initialized before any other drivers **INCLUDING** any application prerequisite drivers.

NOTES:

CONFIGURE_BSP_PREREQUISITE_DRIVERS is typically used by BSPs to configure common infrastructure such as bus controllers or probe for devices.

23.17 Idle Task Configuration

This section defines the IDLE task related configuration parameters supported by <rtems/confdefs.h>.

23.17.1 Specify Application Specific Idle Task Body

CONSTANT: CONFIGURE_IDLE_TASK_BODY

DATA TYPE: method pointer.

RANGE: Undefined or method pointer.

DEFAULT VALUE: By default, this is not defined.

DESCRIPTION:

CONFIGURE_IDLE_TASK_BODY is set to the method name corresponding to the application specific IDLE thread body. If not specified, the BSP or RTEMS default IDLE thread body will be used.

NOTES:

None.

23.17.2 Specify Idle Task Stack Size

CONSTANT: CONFIGURE_IDLE_TASK_STACK_SIZE

DATA TYPE: integer

RANGE: undefined or positive

DEFAULT VALUE: If not specified, the IDLE task will have a stack of the configured minimum stack size.

DESCRIPTION:

CONFIGURE_IDLE_TASK_STACK_SIZE is set to the desired stack size for the IDLE task.

NOTES:

None.

23.17.3 Specify Idle Task Performs Application Initialization

CONSTANT: CONFIGURE_IDLE_TASK_INITIALIZES_APPLICATION

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not the mode of operation and the user is assumed to provide one or more initialization tasks.

DESCRIPTION:

CONFIGURE_IDLE_TASK_INITIALIZES_APPLICATION is set to indicate that the user has configured **NO** user initialization tasks or threads and that the user provided IDLE task will perform application initialization and then transform itself into an IDLE task.

NOTES:

If you use this option be careful, the user IDLE task **CANNOT** block at all during the initialization sequence. Further, once application initialization is complete, it must make itself preemptible and enter an IDLE body loop.

The IDLE task must run at the lowest priority of all tasks in the system.

23.18 Scheduler Algorithm Configuration

This section defines the configuration parameters related to selecting a scheduling algorithm for an application. For the schedulers built into RTEMS, the configuration is straightforward. All that is required is to define the configuration macro which specifies which scheduler you want for in your application. The currently available schedulers are:

The pluggable scheduler interface also enables the user to provide their own scheduling algorithm. If you choose to do this, you must define multiple configuration macros.

23.18.1 Use Deterministic Priority Scheduler

CONSTANT: CONFIGURE_SCHEDULER_PRIORITY

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is the default scheduler and specifying this configuration parameter is redundant.

DESCRIPTION:

The Deterministic Priority Scheduler is the default scheduler in RTEMS for single core applications and is designed for predictable performance under the highest loads. It can block or unblock a thread in a constant amount of time. This scheduler requires a variable amount of memory based upon the number of priorities configured in the system.

NOTES:

This scheduler may be explicitly selected by defining `CONFIGURE_SCHEDULER_PRIORITY` although this is equivalent to the default behavior.

23.18.2 Use Simple Priority Scheduler

CONSTANT: `CONFIGURE_SCHEDULER_SIMPLE`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

When defined, the Simple Priority Scheduler is used at the thread scheduling algorithm. This is an alternative scheduler in RTEMS. It is designed to provide the same task scheduling behaviour as the Deterministic Priority Scheduler while being simpler in implementation and uses less memory for data management. It maintains a single sorted list of all ready threads. Thus blocking or unblocking a thread is not a constant time operation with this scheduler.

This scheduler may be explicitly selected by defining `CONFIGURE_SCHEDULER_SIMPLE`.

NOTES:

This scheduler is appropriate for use in small systems where RAM is limited.

23.18.3 Use Earliest Deadline First Scheduler

CONSTANT: `CONFIGURE_SCHEDULER_EDF`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

The Earliest Deadline First Scheduler (EDF) is an alternative scheduler in RTEMS for single core applications. The EDF schedules tasks with dynamic priorities equal to deadlines. The deadlines are declared using only Rate Monotonic manager which handles periodic behavior. Period is always equal to deadline. If a task does not have any deadline declared or the deadline is cancelled, the task is considered a background task which is scheduled in case no deadline-driven tasks are ready to run. Moreover, multiple background tasks are scheduled

according their priority assigned upon initialization. All ready tasks reside in a single ready queue.

This scheduler may be explicitly selected by defining `CONFIGURE_SCHEDULER_EDF`.

NOTES:

None.

23.18.4 Use Constant Bandwidth Server Scheduler

CONSTANT: `CONFIGURE_SCHEDULER_CBS`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

The Constant Bandwidth Server Scheduler (CBS) is an alternative scheduler in RTEMS for single core applications. The CBS is a budget aware extension of EDF scheduler. The goal of this scheduler is to ensure temporal isolation of tasks. The CBS is equipped with a set of additional rules and provides with an extensive API.

This scheduler may be explicitly selected by defining `CONFIGURE_SCHEDULER_CBS`. See Chapter 30 [Constant Bandwidth Server Scheduler API], page 377 for more details.

NOTES:

None.

23.18.5 Use Simple SMP Priority Scheduler

CONSTANT: `CONFIGURE_SCHEDULER_SIMPLE_SMP`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

The Simple SMP Priority Scheduler is derived from the Simple Priority Scheduler but is capable of scheduling threads across multiple cores. It is designed to provide the same task scheduling behaviour as the Deterministic Priority Scheduler while distributing threads across multiple cores. Being based upon the Simple Priority Scheduler, it also maintains a single sorted list of all ready threads. Thus blocking or unblocking a thread is not a constant time operation with this scheduler.

In addition, when allocating threads to cores, the algorithm is not constant time. This algorithm was not designed with efficiency as a primary design goal. Its primary design goal was to provide an SMP-aware scheduling algorithm that is simple to understand.

In a configuration with SMP enabled at configure time, it may be explicitly selected by defining `CONFIGURE_SCHEDULER_SIMPLE_SMP`.

NOTES:

This scheduler is only available when RTEMS is configured with SMP support enabled.

This scheduler is currently the default in SMP configurations and is only selected when `CONFIGURE_SMP_APPLICATION` is defined.

23.18.6 Configuring a User Provided Scheduler

CONSTANT: `CONFIGURE_SCHEDULER_USER`

DATA TYPE: Entry points for scheduler

RANGE: Undefined or scheduler entry set

DEFAULT VALUE:

DESCRIPTION:

RTEMS allows the application to provide its own task/thread scheduling algorithm. In order to do this, one must define `CONFIGURE_SCHEDULER_USER` to indicate the application provides its own scheduling algorithm. If `CONFIGURE_SCHEDULER_USER` is defined then the following additional macros must be defined:

- `CONFIGURE_MEMORY_FOR_SCHEDULER` must be defined with the amount of memory required as a base amount for the scheduler.
- `CONFIGURE_MEMORY_PER_TASK_FOR_SCHEDULER(_tasks)` must be defined as a formula which computes the amount of memory required based upon the number of tasks configured.

NOTES:

At this time, the mechanics and requirements for writing a new scheduler are evolving and not fully documented. It is recommended that you look at the existing Deterministic Priority Scheduler in `cpukit/score/src/schedulerpriority*.c` for guidance. For guidance on the configuration macros, please examine `cpukit/sapi/include/confdefs.h` for how these are defined for the Deterministic Priority Scheduler.

23.19 SMP Specific Configuration Parameters

This section defines the SMP related system configuration parameters supported by `<rtems/confdefs.h>`. They are only used if the SMP Support (distinct from the Multiprocessing Support) is enabled at configure time using the `--enable-smp` option.

Additionally, this class of Configuration Constants are only applicable if `CONFIGURE_SMP_APPLICATION` is defined.

23.19.1 Specify Application Uses Multiple Cores (is SMP)

CONSTANT: `CONFIGURE_SMP_APPLICATION`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

CONFIGURE_SMP_APPLICATION must be defined if the application is to make use of multiple CPU cores in an SMP target system.

NOTES:

None.

23.19.2 Specify Maximum Processors in SMP System

CONSTANT: CONFIGURE_SMP_MAXIMUM_PROCESSORS

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

CONFIGURE_SMP_MAXIMUM_PROCESSORS must be set to the number of CPU cores in the SMP configuration.

NOTES:

If there are more cores available than configured, the rest will be ignored.

23.20 Device Driver Table

This section defines the configuration parameters related to the automatic generation of a Device Driver Table. As <rtems/confdefs.h> only is aware of a small set of standard device drivers, the generated Device Driver Table is suitable for simple applications with no custom device drivers.

Note that network device drivers are not configured in the Device Driver Table.

23.20.1 Specifying Application Defined Device Driver Table

CONSTANT: CONFIGURE_HAS_OWN_DEVICE_DRIVER_TABLE

DATA TYPE: Array of device drivers.

RANGE: Undefined or array of device drivers.

DEFAULT VALUE: By default, this is not defined indicating the <rtems/confdefs.h> is providing the device driver table.

DESCRIPTION:

CONFIGURE_HAS_OWN_DEVICE_DRIVER_TABLE is defined if the application wishes to provide their own Device Driver Table.

The table must be an array of rtems_driver_address_table entries named Device_drivers.

NOTES:

It is expected that there the application would only rarely need to do this.

23.20.2 Specifying the Maximum Number of Device Drivers

CONSTANT: CONFIGURE_MAXIMUM_DRIVERS

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: By default, this is set to the number of device drivers configured using the CONFIGURE_APPLICATIONS_NEEDS_XXX_DRIVER configuration parameters.

DESCRIPTION:

CONFIGURE_MAXIMUM_DRIVERS is defined as the number of device drivers per node.

NOTES:

If the application will dynamically install device drivers, then this configuration parameter must be larger than the number of statically configured device drivers. Drivers configured using the CONFIGURE_APPLICATIONS_NEEDS_XXX_DRIVER configuration parameters are statically installed.

23.20.3 Specifying Maximum Devices

CONSTANT: CONFIGURE_MAXIMUM_DEVICES

DATA TYPE: integer

RANGE: undefined or positive integer.

DEFAULT VALUE: Unless BSP_MAXIMUM_DEVICES is set by the BSP, the default value for this is set to 4. If overridden by the BSP the value is set to the value specified by the BSP.

DESCRIPTION:

CONFIGURE_MAXIMUM_DEVICES is defined to the number of individual devices that may be registered in the system.

NOTES:

This parameter only impacts the devFS and thus is only used by <rtems/confdefs.h> when CONFIGURE_USE_DEVFS_AS_BASE_FILESYSTEM is specified.

23.20.4 Enable Console Device Driver

CONSTANT:

CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not defined.

DESCRIPTION:

CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER is defined if the application wishes to include the Console Device Driver.

NOTES:

This device driver is responsible for providing standard input and output using */dev/console*.

BSPs should be constructed in a manner that allows `printk()` to work properly without the need for the console driver to be configured.

23.20.5 Enable Clock Driver

CONSTANT: CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not defined.

DESCRIPTION:

CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER is defined if the application wishes to include the Clock Device Driver.

NOTES:

This device driver is responsible for providing a regular interrupt which invokes the `rtems_clock_tick` directive.

If neither the Clock Driver not Benchmark Timer is enabled and the configuration parameter CONFIGURE_APPLICATION_DOES_NOT_NEED_CLOCK_DRIVER is not defined, then a compile time error will occur.

23.20.6 Enable the Benchmark Timer Driver

CONSTANT: CONFIGURE_APPLICATION_NEEDS_TIMER_DRIVER

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not defined.

DESCRIPTION:

CONFIGURE_APPLICATION_NEEDS_TIMER_DRIVER is defined if the application wishes to include the Timer Driver. This device driver is used to benchmark execution times by the RTEMS Timing Test Suites.

NOTES:

If neither the Clock Driver not Benchmark Timer is enabled and the configuration parameter CONFIGURE_APPLICATION_DOES_NOT_NEED_CLOCK_DRIVER is not defined, then a compile time error will occur.

23.20.7 Specify Clock and Benchmark Timer Drivers Are Not Needed

CONSTANT: CONFIGURE_APPLICATION_DOES_NOT_NEED_CLOCK_DRIVER

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

CONFIGURE_APPLICATION_DOES_NOT_NEED_CLOCK_DRIVER is defined when the application does **NOT** want the Clock Device Driver and is **NOT** using the Timer Driver. The inclusion or exclusion of the Clock Driver must be explicit in user applications.

NOTES:

This configuration parameter is intended to prevent the common user error of using the Hello World example as the baseline for an application and leaving out a clock tick source.

23.20.8 Enable Real-Time Clock Driver

CONSTANT: CONFIGURE_APPLICATION_NEEDS_RTC_DRIVER

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not defined.

DESCRIPTION:

CONFIGURE_APPLICATION_NEEDS_RTC_DRIVER is defined if the application wishes to include the Real-Time Clock Driver.

NOTES:

Most BSPs do not include support for a real-time clock. This is because many boards do not include the required hardware.

If this is defined and the BSP does not have this device driver, then the user will get a link time error for an undefined symbol.

23.20.9 Enable the Watchdog Device Driver

CONSTANT: CONFIGURE_APPLICATION_NEEDS_WATCHDOG_DRIVER

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not defined.

DESCRIPTION:

CONFIGURE_APPLICATION_NEEDS_WATCHDOG_DRIVER is defined if the application wishes to include the Watchdog Driver.

NOTES:

Most BSPs do not include support for a watchdog device driver. This is because many boards do not include the required hardware.

If this is defined and the BSP does not have this device driver, then the user will get a link time error for an undefined symbol.

23.20.10 Enable the Graphics Frame Buffer Device Driver

CONSTANT: `CONFIGURE_APPLICATION_NEEDS_FRAME_BUFFER_DRIVER`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not defined.

DESCRIPTION:

`CONFIGURE_APPLICATION_NEEDS_FRAME_BUFFER_DRIVER` is defined if the application wishes to include the BSP's Frame Buffer Device Driver.

NOTES:

Most BSPs do not include support for a Frame Buffer Device Driver. This is because many boards do not include the required hardware.

If this is defined and the BSP does not have this device driver, then the user will get a link time error for an undefined symbol.

23.20.11 Enable Stub Device Driver

CONSTANT: `CONFIGURE_APPLICATION_NEEDS_STUB_DRIVER`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default, this is not defined.

DESCRIPTION:

`CONFIGURE_APPLICATION_NEEDS_STUB_DRIVER` is defined if the application wishes to include the Stub Device Driver.

NOTES:

This device driver simply provides entry points that return successful and is primarily a test fixture. It is supported by all BSPs.

23.20.12 Specify Application Prerequisite Device Drivers

CONSTANT: `CONFIGURE_APPLICATION_PREREQUISITE_DRIVERS`

DATA TYPE: device driver entry structures

RANGE: Undefined or set of device driver entry structures

DEFAULT VALUE: By default,this is not defined.

DESCRIPTION:

CONFIGURE_APPLICATION_PREREQUISITE_DRIVERS is defined if the application has device drivers it needs to include in the Device Driver Table. This should be defined to the set of device driver entries that will be placed in the table at the **FRONT** of the Device Driver Table and initialized before any other drivers **EXCEPT** any BSP prerequisite drivers.

NOTES:

In some cases, it is used by System On Chip BSPs to support peripheral buses beyond those normally found on the System On Chip. For example, this is used by one RTEMS system which has implemented a SPARC/ERC32 based board with VMEBus. The VMEBus Controller initialization is performed by a device driver configured via this configuration parameter.

23.20.13 Specify Extra Application Device Drivers

CONSTANT: CONFIGURE_APPLICATION_EXTRA_DRIVERS

DATA TYPE: device driver entry structures

RANGE: Undefined or set of device driver entry structures

DEFAULT VALUE: By default,this is not defined.

DESCRIPTION:

CONFIGURE_APPLICATION_EXTRA_DRIVERS is defined if the application has device drivers it needs to include in the Device Driver Table. This should be defined to the set of device driver entries that will be placed in the table at the **END** of the Device Driver Table.

NOTES:

None.

23.20.14 Enable /dev/null Device Driver

CONSTANT: CONFIGURE_APPLICATION_NEEDS_NULL_DRIVER

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default,this is not defined.

DESCRIPTION:

This configuration variable is specified to enable */dev/null* device driver.

NOTES:

This device driver is supported by all BSPs.

23.20.15 Enable /dev/zero Device Driver

CONSTANT: CONFIGURE_APPLICATION_NEEDS_ZERO_DRIVER

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: By default,this is not defined.

DESCRIPTION:

This configuration variable is specified to enable */dev/zero* device driver.

NOTES:

This device driver is supported by all BSPs.

23.21 Multiprocessing Configuration

This section defines the multiprocessing related system configuration parameters supported by <rtems/confdefs.h>. They are only used if the Multiprocessing Support (distinct from the SMP support) is enabled at configure time using the --enable-multiprocessing option.

Additionally, this class of Configuration Constants are only applicable if CONFIGURE_MP_APPLICATION is defined.

23.21.1 Specify Application Will Use Multiprocessing

CONSTANT: CONFIGURE_MP_APPLICATION

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

This configuration parameter must be defined to indicate that the application intends to be part of a multiprocessing configuration. Additional configuration parameters are assumed to be provided.

NOTES:

This has no impact unless RTEMS was configured and built using the --enable-multiprocessing option.

23.21.2 Configure Node Number in Multiprocessor Configuration

CONSTANT: CONFIGURE_MP_NODE_NUMBER

DATA TYPE: integer

RANGE: positive integer

DEFAULT VALUE: If not defined, it is set to NODE_NUMBER which is assumed to be set by the compilation environment.

DESCRIPTION:

CONFIGURE_MP_NODE_NUMBER is the node number of this node in a multiprocessor system.

NOTES:

In the RTEMS Multiprocessing Test Suite, the node number is derived from the Makefile variable NODE_NUMBER. The same code is compiled with the NODE_NUMBER set to different values. The test programs behave differently based upon their node number.

23.21.3 Configure Maximum Node in Multiprocessor Configuration

CONSTANT: CONFIGURE_MP_MAXIMUM_NODES

DATA TYPE: integer

RANGE: positive

DEFAULT VALUE: The default is two (2).

DESCRIPTION:

CONFIGURE_MP_MAXIMUM_NODES is the maximum number of nodes in a multiprocessor system.

NOTES:

None.

23.21.4 Configure Maximum Global Objects in Multiprocessor Configuration

CONSTANT: CONFIGURE_MP_MAXIMUM_GLOBAL_OBJECTS

DATA TYPE: integer

RANGE: positive

DEFAULT VALUE: The default is 32.

DESCRIPTION:

CONFIGURE_MP_MAXIMUM_GLOBAL_OBJECTS is the maximum number of concurrently active global objects in a multiprocessor system.

NOTES:

This value corresponds to the total number of objects which can be created with the RTEMS_GLOBAL attribute.

23.21.5 Configure Maximum Proxies in Multiprocessor Configuration

CONSTANT: CONFIGURE_MP_MAXIMUM_PROXIES

DATA TYPE: integer

RANGE: undefined or positive

DEFAULT VALUE: The default is 32.

DESCRIPTION:

CONFIGURE_MP_MAXIMUM_PROXIES is the maximum number of concurrently active thread/task proxies on this node in a multiprocessor system.

NOTES:

Since a proxy is used to represent a remote task/thread which is blocking on this node. This configuration parameter reflects the maximum number of remote tasks/threads which can be blocked on objects on this node.

See Section 24.2.5 [Multiprocessing Manager Proxies], page 303 for more details.

23.21.6 Configure MPCI in Multiprocessor Configuration

CONSTANT: CONFIGURE_MP_MPCI_TABLE_POINTER

DATA TYPE: pointer to rtems_mpci_table

RANGE: undefined or valid pointer

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

CONFIGURE_MP_MPCI_TABLE_POINTER is the pointer to the MPCI Configuration Table. The default value of this field is &MPCI_table.

NOTES:

RTEMS provides a Shared Memory MPCI Device Driver which can be used on any Multiprocessor System assuming the BSP provides the proper set of supporting methods.

23.21.7 Do Not Generate Multiprocessor Configuration Table

CONSTANT: CONFIGURE_HAS_OWN_MULTIPROCESSING_TABLE

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

CONFIGURE_HAS_OWN_MULTIPROCESSING_TABLE is defined if the application wishes to provide their own Multiprocessing Configuration Table. The generated table is named Multiprocessing_configuration.

NOTES:

This is a configuration parameter which is very unlikely to be used by an application. If you find yourself wanting to use it in an application, please reconsider and discuss this on the RTEMS Users mailing list.

23.22 Ada Tasks

This section defines the system configuration parameters supported by `<rtems/confdefs.h>` related to configuring RTEMS to support a task using Ada tasking with GNAT/RTEMS.

These configuration parameters are only available when RTEMS is built with the `--enable-ada` configure option and the application specifies `CONFIGURE_GNAT_RTEMS`.

Additionally RTEMS includes an Ada language binding to the Classic API which has a test suite. This test suite is enabled only when `--enable-tests` and `--enable-expada` are specified on the configure command.

23.22.1 Specify Application Includes Ada Code

CONSTANT: `CONFIGURE_GNAT_RTEMS`

DATA TYPE: Boolean feature macro.

RANGE: Defined or undefined.

DEFAULT VALUE: This is not defined by default.

DESCRIPTION:

`CONFIGURE_GNAT_RTEMS` is defined to inform RTEMS that the GNAT Ada run-time is to be used by the application.

NOTES:

This configuration parameter is critical as it makes `<rtems/confdefs.h>` configure the resources (POSIX API Threads, Mutexes, Condition Variables, and Keys) used implicitly by the GNAT run-time.

23.22.2 Specify the Maximum Number of Ada Tasks.

CONSTANT: `CONFIGURE_MAXIMUM_ADA_TASKS`

DATA TYPE: integer

RANGE: undefined or positive

DEFAULT VALUE: By default, when `CONFIGURE_GNAT_RTEMS` is defined, this is set to 20.

DESCRIPTION:

`CONFIGURE_MAXIMUM_ADA_TASKS` is the number of Ada tasks that can be concurrently active in the system.

NOTES:

None.

23.22.3 Specify the Maximum Fake Ada Tasks

CONSTANT:

DATA TYPE: integer

RANGE: zero or positive

DEFAULT VALUE: By default, this is undefined which implies zero (0) *fake* Ada Tasks.

DESCRIPTION:

`CONFIGURE_MAXIMUM_FAKE_ADA_TASKS` is the number of *fake* Ada tasks that can be concurrently active in the system. A *fake* Ada task is a non-Ada task that makes calls back into Ada code and thus implicitly uses the Ada run-time.

NOTES:

None.

23.23 Configuration Data Structures

It is recommended that applications be configured using `<rtems/confdefs.h>` as it is simpler and insulates applications from changes in the underlying data structures. However, it is sometimes important to understand the data structures that are automatically filled in by the configuration parameters. This section describes the primary configuration data structures.

If the user wishes to see the details of a particular data structure, they are are advised to look at the source code. After all, that is one of the advantages of RTEMS.