3

Initialization Manager

3.1 Introduction

The **initialization manager** is responsible for initiating and shutting down **RTEMS**. Initiating **RTEMS** involves creating and starting all configured initialization tasks, and for invoking the initialization routine for each user-supplied device driver. In a multiprocessor configuration, this manager also initializes the interprocessor communications layer. The directives provided by the initialization manager are:

Name	Directive Description
initialize_executive	Initialize RTEMS
initialize_executive_early	Initialize RTEMS and do NOT Start Multitasking
initialize_executive_late	Complete Initialization and Start Multitasking
shutdown_executive	Shutdown RTEMS

3.2 Background

3.2.1 Initialization Tasks

Initialization task(s) are the mechanism by which **RTEMS** transfers initial control to the user's application. Initialization tasks differ from other application tasks in that they are defined in the **User Initialization Tasks Table** and automatically created and started by **RTEMS** as part of its initialization sequence. Since the initialization tasks are scheduled using the same algorithm as all other

RTEMS tasks, they must be configured at a priority and mode which will insure that they will complete execution before other application tasks execute. Although there is no upper limit on the number of initialization tasks, an application is required to define at least one.

A typical initialization task will create and start the static set of application tasks. It may also create any other objects used by the application. Initialization tasks which only perform initialization should delete themselves upon completion to free resources for other tasks. Initialization tasks may transform themselves into a "normal" application task. This transformation typically involves changing priority and execution mode. **RTEMS does not automatically delete the initialization tasks.**

3.2.2 The System Initialization Task

The **System Initialization Task** is responsible for initializing all device drivers. As a result, this task has a higher priority than all other tasks to insure that no application tasks executes until all device drivers are initialized. After device initialization in a single processor system, this task will delete itself.

The **System Initialization Task** must have enough stack space to successfully execute the initialization routines for all device drivers and, in multiprocessor configurations, the **Multiprocessor Communications Interface Layer** initialization routine. The **CPU Configuration Table** contains a field which allows the application or BSP to increase the default amount of stack space allocated for this task.

In multiprocessor configurations, the **System Initialization Task** does not delete itself after initializing the device drivers. Instead it transforms itself into the **Multiprocessing Server** which initializes the **Multiprocessor Communications Interface Layer**, verifies multiprocessor system consistency, and processes all requests from remote nodes.

3.2.3 The Idle Task

The **Idle Task** is the lowest priority task in a system and executes only when no other task is ready to execute. This task consists of an infinite loop and will be preempted when any other task is made ready to execute.

3.2.4 Initialization Manager Failure

The **fatal_error_occurred** directive will be called from **initialize_executive** for any of the following reasons:

- *If no user initialization tasks are configured. At least one initialization task must be configured to allow **RTEMS** to pass control to the application at the end of the executive initialization sequence.*
- *If a **CPU Dependent Information Table** is required by the target processor and **NULL** is passed to **initialize_executive**.*
- *If the starting address of the **RTEMS RAM Workspace**, supplied by the application in the **Configuration Table**, is **NULL** or is not aligned on a four-byte boundary.*
- *If the size of the **RTEMS RAM Workspace** is not large enough to initialize and configure the system.*
- *If multiprocessing is configured and the **node** entry in the **Multiprocessor Configuration Table** is not between one and the **maximum_nodes** entry.*
- *If any of the user initialization tasks cannot be created or started successfully.*

3.3 Operations

3.3.1 Initializing RTEMS

The **initialize_executive** directive is called by the board support package at the completion of its initialization sequence. **RTEMS** assumes that the board support package successfully completed its initialization activities. The **initialize_executive** directive completes the initialization sequence by performing the following actions:

- *Initializing internal **RTEMS** variables;*
- *Allocating system resources;*
- *Creating and starting the **System Initialization Task**;*
- *Creating and starting the **Idle Task**;*
- *Creating and starting the user initialization task(s); and*
- *Initiating multitasking.*

This directive **MUST** be called before any other **RTEMS** directives. The effect of calling any **RTEMS** directives before **initialize_executive** is unpredictable. Many of **RTEMS** actions during initialization are based upon the contents of the **Configuration Table** and **CPU Dependent Information Table**. For more

information regarding the format and contents of these tables, please refer to the chapter **Configuring a System**.

The final step in the initialization sequence is the initiation of multitasking. When the scheduler and dispatcher are enabled, the highest priority, ready task will be dispatched to run. Control will not be returned to the board support package after multitasking is enabled until **shutdown_executive** the directive is called.

The **initialize_executive** directive provides a conceptually simple way to initialize **RTEMS**. However, in certain cases, this mechanism cannot be used. The **initialize_executive_early** and **initialize_executive_late** directives are provided as an alternative mechanism for initializing **RTEMS**. The **initialize_executive_early** directive returns to the caller **BEFORE** initiating multitasking. The **initialize_executive_late** directive is invoked to start multitasking. It is critical that only one of the **RTEMS** initialization sequences be used in an application.

3.3.2 Shutting Down RTEMS

The **shutdown_executive** directive is invoked by the application to end multitasking and return control to the board support package. The board support package resumes execution at the code immediately following the invocation of the **initialize_executive** directive.

3.4 Directives

This section details the initialization manager's directives. A subsection is dedicated to each of this manager's directives and describes the calling sequence, related constants, usage, and status codes.

3.4.1 INITIALIZE_EXECUTIVE - Initialize RTEMS

CALLING SEQUENCE:

```
function EXECUTIVE_EARLY (
  CONFIGURATION_TABLE   : in     CONFIGURATION.TABLE;
  CPU_TABLE             : in     CPU.CONFIGURATION_TABLE;
  TASK_TABLE            : in     CONFIGURATION.INITIALIZATION_TASKS_TABLE;
  DRIVER_TABLE          : in     CONFIGURATION.DRIVER_ADDRESS_TABLE;
  EXTENSION_TABLE       : in     CONFIGURATION.EXTENSION_TABLE;
  MULTIPROCESSING_TABLE : in     CONFIGURATION.MULTIPROCESSING_TABLE;
  MPCI_TABLE            : in     CONFIGURATION.MPCI_TABLE
) return INTERRUPTS.LEVEL;
```

DIRECTIVE STATUS CODES:

NONE

DESCRIPTION:

This directive is called when the board support package has completed its initialization to allow **RTEMS** to initialize the application environment based upon the information in the **Configuration Table**, **CPU Dependent Information Table**, **User Initialization Tasks Table**, **Device Driver Table**, **User Extension Table**, **Multiprocessor Configuration Table**, and the **Multiprocessor Communications Interface (MPCI) Table**. This directive starts multitasking and does not return to the caller until the **shutdown_executive** directive is invoked.

NOTES:

This directive **MUST** be the first **RTEMS** directive called and it **DOES NOT RETURN** to the caller until the **shutdown_executive** is invoked.

This directive causes all nodes in the system to verify that certain configuration parameters are the same as those of the local node. If an inconsistency is detected, then a fatal error is generated.

The application must use only one of the two initialization sequences: **initialize_executive** or **initialize_executive_early** and **initialize_executive_late**. The **initialize_executive** directive is logically equivalent to invoking **initialize_executive_early** and **initialize_executive_late** with no intervening actions.

3.4.2 INITIALIZE_EXECUTIVE_EARLY - Initialize RTEMS and do NOT Start Multitasking

CALLING SEQUENCE:

```
function EXECUTIVE_EARLY (
  CONFIGURATION_TABLE   : in     CONFIGURATION.TABLE;
  CPU_TABLE             : in     CPU.CONFIGURATION_TABLE;
  TASK_TABLE            : in     CONFIGURATION.INITIALIZATION_TASKS_TABLE;
  DRIVER_TABLE          : in     CONFIGURATION.DRIVER_ADDRESS_TABLE;
  EXTENSION_TABLE       : in     CONFIGURATION.EXTENSION_TABLE;
  MULTIPROCESSING_TABLE : in     CONFIGURATION.MULTIPROCESSING_TABLE;
  MPCI_TABLE            : in     CONFIGURATION.MPCI_TABLE
) return INTERRUPTS.LEVEL;
```

DIRECTIVE STATUS CODES:

NONE

DESCRIPTION:

This directive is called when the board support package has completed its initialization to allow **RTEMS** to initialize the application environment based upon the information in the **Configuration Table**, **CPU Dependent Information Table**, **User Initialization Tasks Table**, **Device Driver Table**, **User Extension Table**, **Multiprocessor Configuration Table**, and the **Multiprocessor Communications Interface (MPCI) Table**. This directive returns to the caller after completing the basic **RTEMS** initialization but before multitasking is initiated. The interrupt level in place when the directive is invoked is returned to the caller. This interrupt level should be the same one passed to **initialize_executive_late**.

NOTES:

The application must use only one of the two initialization sequences: **initialize_executive** or **initialize_executive_early** and **initialize_executive_late**.

3.4.3 INITIALIZE_EXECUTIVE_LATE - Complete Initialization and Start Multitasking

CALLING SEQUENCE:

```
procedure EXECUTIVE_LATE (
   BSP_LEVEL : in     INTERRUPTS.LEVEL
);
```

DIRECTIVE STATUS CODES:

NONE

DESCRIPTION:

This directive is called after the **initialize_executive_early** directive has been called to complete the **RTEMS** initialization sequence and initiate multitasking. The interrupt level returned by the **initialize_executive_early** directive should be in **bsp_level** and this value is restored as part of this directive returning to the caller after the **shutdown_executive** directive is invoked.

NOTES:

This directive **MUST** be the second **RTEMS** directive called and it **DOES NOT RETURN** to the caller until the **shutdown_executive** is invoked.

This directive causes all nodes in the system to verify that certain configuration parameters are the same as those of the local node. If an inconsistency is detected, then a fatal error is generated.

The application must use only one of the two initialization sequences: **initialize_executive** or **initialize_executive_early** and **initialize_executive_late**.

3.4.4 SHUTDOWN_EXECUTIVE - Shutdown RTEMS

CALLING SEQUENCE:

```
procedure shutdown_executive (
   result : in     rtems.unsigned32
);
```

DIRECTIVE STATUS CODES:

NONE

DESCRIPTION:

This directive is called when the application wishes to shutdown **RTEMS** and return control to the board support package. The board support package resumes execution at the code immediately following the invocation of the **initialize_executive** directive.

NOTES:

This directive **MUST** be the last **RTEMS** directive invoked by an application and it **DOES NOT RETURN** to the caller.

This directive should not be invoked until the executive has successfully completed initialization.