20

User Extensions Manager

20.1 Introduction

The **RTEMS** User Extensions Manager allows the application developer to augment the executive by allowing them to supply extension routines which are invoked at critical system events. The directives provided by the user extensions manager are:

Name	Directive Description
extension_create	Create an extension set
extension_ident	Get ID of an extension set
extension_delete	Delete an extension set

20.2 Background

User extension routines are invoked when the following system events occur:

- *Task creation*
- *Task initiation*
- *Task reinitiation*
- *Task deletion*
- *Task context switch*
- *Task begins to execute*
- *Task exits*
- *Fatal error detection*

These extensions are invoked as a subprogram with arguments that are appropriate to the system event.

20.2.1 Extension Sets

An extension set is defined as a set of routines which are invoked at each of the critical system events at which user extension routines are invoked. Together a set of these routines typically perform a specific functionality such as performance montoring or debugger support. **RTEMS** is informed of the entry points which constitute an extension set via the following record:

```
type extension_table is
   record
      task_create       : rtems.addresses;  -- task create extension
      task_start        : rtems.addresses;  -- task start extension
      task_restart      : rtems.addresses;  -- task restart extension
      task_delete       : rtems.addresses;  -- task delete extension
      task_switch       : rtems.addresses;  -- task switch extension
      task_begin        : rtems.addresses;  -- task begin extension
      task_exitted      : rtems.addresses;  -- task exitted extension
      fatal_error       : rtems.addresses;  -- fatal error extension
   end record;
```

RTEMS allows the user to have multiple extension sets active at the same time. First, a single static extension set may be defined as the application's **User Extension Table** which is included as part of the **Configuration Table**. This extension set is active for the entire life of the system and may not be deleted. This extension set is especially important because it is the only way the application can provided a **FATAL** error extension which is invoked if **RTEMS** fails during the **initialize_executive** directive. The static extension set is optional and may be configured as **NULL** if no static extension set is required.

Second, the user can install dynamic extensions using the **extension_create** directive. These extensions are **RTEMS** objects in that they have a name, an ID, and can be dynamically created and deleted. In contrast to the static extension set, these extensions can only be created and installed after the **initialize_executive** directive successfully completes execution. Dynamic extensions are useful for encapsulating the functionality of an extension set. For example, the application could use extensions to manage a special coprocessor, do performance monitoring, and to do stack bounds checking. Each of these extension sets could be written and installed independently of the others.

All user extensions are optional and **RTEMS** places no naming restrictions on the user.

20.2.2 TCB Extension Area

RTEMS provides for a user-defined data area to be linked to each task's control block. This data area is an extension of the TCB and can be used to store additional data required by one or more of the user's extension functions. It is also

possible for a user extension to utilize the notepad locations associated with each task.

The TCB extension is a field in the TCB which can be set by the user to access an area of RAM. This allows an application to augment the TCB with user-defined information. For example, an application could implement task profiling by storing timing statistics in the TCB's extended memory area. When a task context switch is being executed, the TASK_SWITCH extension could read a real-time clock to calculate how long the task being swapped out has run as well as timestamp the starting time for the task being swapped in.

If used, the extended memory area for the TCB should be allocated and the TCB extension pointer should be set at the time the task is created or started by either the TASK_CREATE or TASK_START extension. The application is responsible for managing this extended memory area for the TCBs. The memory may be reinitialized by the TASK_RESTART extension and should be deallocated by the TASK_DELETE extension when the task is deleted. Since the TCB extension buffers would most likely be of a fixed size, the **RTEMS** partition manager could be used to manage the application's extended memory area. The application could create a partition of fixed size TCB extension buffers and use the partition manager's allocation and deallocation directives to obtain and release the extension buffers.

20.2.3 Extensions

The sections that follow will contain a description of each extension. Each section will contain a specification of a subprogram with the appropriate calling sequence for the corresponding extension. The names given for the Ada subprogram and its arguments are all defined by the user. The names used in the examples were arbitrarily chosen and impose no naming conventions on the user.

20.2.3.1 TASK_CREATE Extension

The TASK_CREATE extension directly corresponds to the **task_create** directive. If this extension is defined in any static or dynamic extension set and a task is being created, then the extension routine will automatically be invoked by **RTEMS**. The extension should have a specification similar to the following:

```
procedure user_task_create(
   current_task : in     rtems.tcb_pointer;
   new_task     : in     rtems.tcb_pointer
);
```

where **current_task** can be used to access the TCB for the currently executing task, and **new_task** can be used to access the TCB for the new task being created.

This extension is invoked from the **task_create** directive after **new_task** has been completely initialized, but before it is placed on a ready TCB chain.

20.2.3.2 TASK_START Extension

The TASK_START extension directly corresponds to the **task_start** directive. If this extension is defined in any static or dynamic extension set and a task is being started, then the extension routine will automatically be invoked by **RTEMS**. The extension should have a specification similar to the following:

```
procedure user_task_start(
   current_task : in     rtems.tcb_pointer;
   started_task : in     rtems.tcb_pointer
);
```

where **current_task** can be used to access the TCB for the currently executing task, and **started_task** can be used to access the TCB for the dormant task being started. This extension is invoked from the **task_start** directive after **started_task** has been made ready to start execution, but before it is placed on a ready TCB chain.

20.2.3.3 TASK_RESTART Extension

The TASK_RESTART extension directly corresponds to the **task_restart** directive. If this extension is defined in any static or dynamic extension set and a task is being restarted, then the extension should have a specification similar to the following:

```
procedure user_task_restart(
   current_task   : in     rtems.tcb_pointer;
   restarted_task : in     rtems.tcb_pointer
);
```

where **current_task** can be used to access the TCB for the currently executing task, and **restarted_task** can be used to access the TCB for the task being restarted. This extension is invoked from the **task_restart** directive after **restarted_task** has been made ready to start execution, but before it is placed on a ready TCB chain.

20.2.3.4 TASK_DELETE Extension

The TASK_DELETE extension is associated with the **task_delete** directive. If this extension is defined in any static or dynamic extension set and a task is being deleted, then the extension routine will automatically be invoked by **RTEMS**. The extension should have a specification similar to the following:

```
procedure user_task_delete(
   current_task : in     rtems.tcb_pointer;
   deleted_task : in     rtems.tcb_pointer
);
```

where **current_task** can be used to access the TCB for the currently executing task, and **deleted_task** can be used to access the TCB for the task being deleted. This extension is invoked from the **task_delete** directive after the TCB has been removed from a ready TCB chain, but before all its resources including the TCB have been returned to their respective free pools. This extension should not call any **RTEMS** directives if a task is deleting itself (**current_task** is equal to **deleted_task**).

20.2.3.5 TASK_SWITCH Extension

The TASK_SWITCH extension corresponds to a task context switch. If this extension is defined in any static or dynamic extension set and a task context switch is in progress, then the extension routine will automatically be invoked by **RTEMS**. The extension should have a specification similar to the following:

```
procedure user_task_switch(
   current_task : in     rtems.tcb_pointer;
   heir_task    : in     rtems.tcb_pointer
);
```

where **current_task** can be used to access the TCB for the task that is being swapped out, and **heir_task** can be used to access the TCB for the task being swapped in. This extension is invoked from **RTEMS**' dispatcher routine after the **current_task** context has been saved, but before the **heir_task** context has been restored. This extension should not call any **RTEMS** directives.

20.2.3.6 TASK_BEGIN Extension

The TASK_BEGIN extension is invoked when a task begins execution. It is invoked immediately before the body of the starting procedure and executes in the context in the task. This user extension have a specification similar to the following:

```
procedure user_task_begin(
   current_task : in     rtems.tcb_pointer
);
```

where **current_task** can be used to access the TCB for the currently executing task which has begun. The distinction between the TASK_BEGIN and TASK_START extension is that the TASK_BEGIN extension is executed in the context of the actual task while the TASK_START extension is executed in the context of the task performing the **task_start** directive. For most extensions, this is not a critical distinction.

20.2.3.7 TASK_EXITTED Extension

The TASK_EXITTED extension is invoked when a task exits the body of the starting procedure by either an implicit or explicit return statement. This user extension have a specification similar to the following:

```
procedure user_task_exitted(
   current_task : in     rtems.tcb_pointer
);
```

where **current_task** can be used to access the TCB for the currently executing task which has just exitted.

Although exiting of task is often considered to be a fatal error, this extension allows recovery by either restarting or deleting the exiting task. If the user does not wish to recover, then a fatal error may be reported. If the user does not provide a TASK_EXITTED extension or the provided handler returns control to **RTEMS**, then the **RTEMS** default handler will be used. This default handler invokes the directive **fatal_error_occurred** with the **TASK_EXITTED** directive status.

20.2.3.8 FATAL Error Extension

The FATAL error extension is associated with the **fatal_error_occurred** directive. If this extension is defined in any static or dynamic extension set and the **fatal_error_occurred** directive has been invoked, then this extension will be called. This extension should have a specification similar to the following:

```
procedure user_fatal_error_extension(
   the_error : in     rtems.unsigned32
);
```

where **the_error** is the error code passed to the **fatal_error_occurred** directive. This extension is invoked from the **fatal_error_occurred** directive.

If defined, the user's FATAL error extension is invoked before **RTEMS**' default fatal error routine is invoked and the processor is stopped. For example, this extension could be used to pass control to a debugger when a fatal error occurs. This extension should not call any **RTEMS** directives.

20.2.4 Order of Invocation

When one of the critical system events occur, the user extensions are invoked in either "forward" or "reverse" order. Forward order indicates that the static extension set is invoked followed by the dynamic extension sets in the order in which they were created. Reverse order means that the dynamic extension sets are invoked in the opposite of the order in which they were created followed by the static extension set. By invoking the extension sets in this order, extensions can be

built upon one another. At the following system events, the extensions are invoked in forward order:

- *Task creation*
- *Task initiation*
- *Task reinitiation*
- *Task deletion*
- *Task context switch*
- *Task begins to execute*

At the following system events, the extensions are invoked in reverse order:

- *Task deletion*
- *Fatal error detection*

At these system events, the extensions are invoked in reverse order to insure that if an extension set is built upon another, the more complicated extension is invoked before the extension set it is built upon. For example, by invoking the static extension set last it is known that the "system" fatal error extension will be the last fatal error extension executed. Another example is use of the task delete extension by the Standard C Library. Extension sets which are installed after the Standard C Library will operate correctly even if they utilize the C Library because the C Library's TASK_DELETE extension is invoked **after** that of the other extensions.

20.3 Operations

20.3.1 Creating an Extension Set

The **extension_create** directive creates and installs an extension set by allocating a **Extension Set Control Block** (ESCB), assigning the extension set a user-specified name, and assigning it an extension set ID. Newly created extension sets are immediately installed and are invoked upon the next system even supporting an extension.

20.3.2 Obtaining Extension Set IDs

When an extension set is created, **RTEMS** generates a unique extension set ID and assigns it to the created extension set until it is deleted. The extension ID may be obtained by either of two methods. First, as the result of an invocation of the **extension_create** directive, the extension set ID is stored in a user provided location. Second, the extension set ID may be obtained later using the **extension_ident** directive. The extension set ID is used by other directives to manipulate this extension set.

20.3.3 Deleting an Extension Set

The **extension_delete** directive is used to delete an extension set. The extension set's control block is returned to the ESCB free list when it is deleted. An extension set can be deleted by a task other than the task which created the extension set. Any subsequent references to the extension's name and ID are invalid.

20.4 Directives

This section details the user extension 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.

20.4.1 EXTENSION_CREATE - Create a extension set

CALLING SEQUENCE:

```
procedure extension_create (
  name   : in     rtems.name;
  table  : in     rtems.extensions_table;
  id     :    out rtems.id;
  status :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	extension set created successfully
INVALID_NAME	invalid extension set name
TOO_MANY	too many extension sets created

DESCRIPTION:

This directive creates a extension set. The assigned extension set id is returned in **id**. This id is used to access the extension set with other user extension manager directives. For control and maintenance of the extension set, **RTEMS** allocates an ESCB from the local ESCB free pool and initializes it.

NOTES:

This directive will not cause the calling task to be preempted.

20.4.2 EXTENSION_IDENT - Get ID of a extension set

CALLING SEQUENCE:

```
procedure extension_ident (
  name   : in     rtems.name;
  id     :    out rtems.id;
  status :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	extension set identified successfully
INVALID_NAME	extension set name not found

DESCRIPTION:

This directive obtains the extension set **id** associated with the extension set **name** to be acquired. If the extension set name is not unique, then the extension set id will match one of the extension sets with that name. However, this extension set id is not guaranteed to correspond to the desired extension set. The extension set id is used to access this extension set in other extension set related directives.

NOTES:

This directive will not cause the running task to be preempted.

20.4.3 EXTENSION_DELETE - Delete a extension set

CALLING SEQUENCE:

```
procedure extension_delete (
  id     :    out rtems.id;
  status :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	extension set deleted successfully
INVALID_ID	invalid extension set id

DESCRIPTION:

This directive deletes the extension set specified by **id**. If the extension set is running, it is automatically canceled. The ESCB for the deleted extension set is reclaimed by **RTEMS**.

NOTES:

This directive will not cause the running task to be preempted.

A extension set can be deleted by a task other than the task which created the extension set.

NOTES:

This directive will not cause the running task to be preempted.