CEC Continuing Education Center

DesignNews

Embedded Software Design Techniques

DAY 5 : System Configuration Management Techniques

Sponsored by

Digi-Key ELECTRONICS

BENINGO EMBEDDED GROUP

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.
- Submit questions for the lecturer using the Q&A widget. They will follow-up after the lecture portion concludes.

Course Sessions

- Software Architectures 101
- Designing RTOS-based Applications
- Architecture Verification Techniques
- Designing Quality into Embedded Systems
- **Software Configuration Management Techniques**

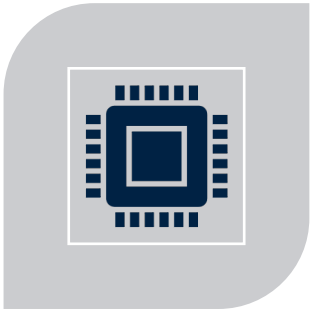
1 Flexible Task Creation Pattern

Task information and creation is often scattered through-out application code. This is not just annoying, but creates software that is difficult to scale, maintain and debug.

Flexible Task Creation Pattern

There are several best practices developers should follow when creating tasks:

TASKS SHOULD ALL EXIST IN THEIR OWN SEPARATE CODE MODULE



TASK CODE CAN BE MADE PRIVATE USING STATIC AND THEN USING A TASK_NAMEINIT FUNCTION

TASK CODE CAN BE MADE PUBLIC USING EXTERN AND THEN CREATED USING A CONFIGURATION TABLE.

TASK INITIALIZATION PARAMETERS SHOULD ALL EXIST IN ONE EASY TO FIND PLACE.

Flexible Task Creation Pattern

Tasks have several common config values

- Task Priority
- Stack depth
- Period (for periodic tasks)

Benefits to a task configuration file:

- All task configuration in a single place
- Human readable name
- Easy to make system level changes

Task_config.h

```
#define TASK_SENSOR_PRIORITY          (17U)
#define TASK_TELEMETRY_PRIORITY       (17U)
#define TASK_LED2BLINK_PRIORITY       (17U)

#define TASK_SENSOR_STACK_DEPTH       (2048U)
#define TASK_TELEMETRY_STACK_DEPTH    (2048U)
#define TASK_LED2BLINK_STACK_DEPTH    (256U)

#define TASK_SENSOR_PERIOD_MS         (25U)
#define TASK_TELEMETRY_PERIOD_MS      (100U)
#define TASK_ LED2BLINK _PERIOD_MS    (100U)
```

Flexible Task Creation Pattern

A **task configuration table** is made up of a developer defined structure with all the parameters necessary to create a task.

Benefits:

- A single structure to organize all task creation.
- An array can be created to hold all task creation parameters
- Any changes to tasks can be done in one place. (Change value, add/remove task).
- A single loop can initialize all system tasks

```
// Task configuration structure
typedef struct
{
    TaskFunction_t TaskCodePtr;
    const char * const TaskName;
    const configSTACK_DEPTH_TYPE StackDepth;
    void * const ParametersPtr;
    UBaseType_t TaskPriority;
    TaskHandle_t * const TaskHandle;
}TaskInitParams_t;
```

Flexible Task Creation Pattern

```
TaskInitParams_t TaskInitParameters[] =
{
   // Pointer to the Task function    , Task String Name, The task stack depth              , Parameter Pointer, Task priority                  , Task Handle
   {(TaskFunction_t)Task_Telemetry, "Task_Telemetry", TASK_TELEMETRY_STACK_DEPTH, NULL      , TASK_TELEMETRY_PRIORITY, NULL      },
   {(TaskFunction_t)Task_Led2Blink, "Task_Led2Blink", TASK_LED2BLINK_STACK_DEPTH , NULL      , TASK_LED2BLINK_PRIORITY , NULL      }
   {(TaskFunction_t)Task_Led3Blink, "Task_Led3Blink", TASK_LED3BLINK_STACK_DEPTH , NULL      , TASK_LED3BLINK_PRIORITY , NULL      }
   {(TaskFunction_t)Task_Sensors   , "Task_Sensors"  , TASK_SENSORS_STACK_DEPTH   , NULL      , TASK_SENSORS_PRIORITY   , NULL      },
};
```

Flexible Task Creation Pattern

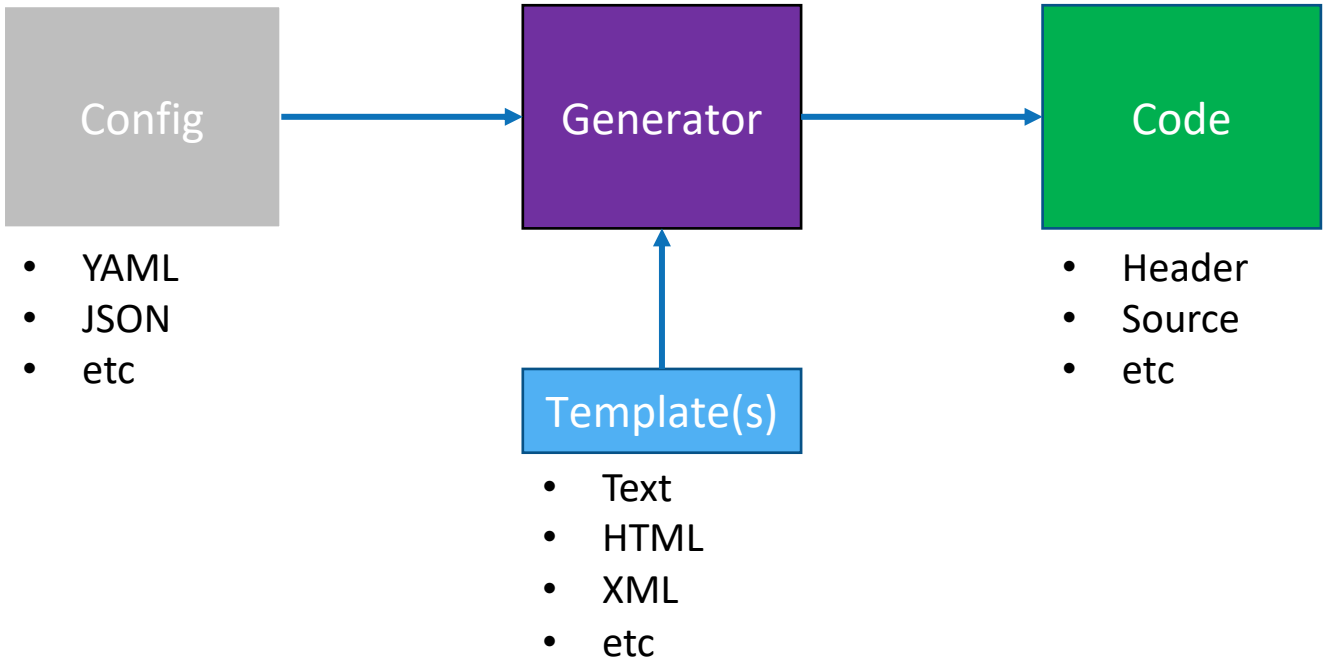
The for loop to manage creating the tasks might look like:

```
for(TaskCount = 0; TaskCount < TasksToCreate; TaskCount++)
{
   xTaskCreate(TaskInitParameters[TaskCount].TaskCodePtr,
               TaskInitParameters[TaskCount].TaskName,
               TaskInitParameters[TaskCount].StackDepth,
               TaskInitParameters[TaskCount].ParametersPtr,
               TaskInitParameters[TaskCount].TaskPriority,
               TaskInitParameters[TaskCount].TaskHandle);
}
```

How do you create tasks in your RTOS applications?
- Scatter tasks all over the application
- Initialize them individually in main
- Use a configuration table with an initializing function
- Other

2 Automating Configuration

Automating Configuration



Do you manually or automatically generate your configuration?
- Automatically
- Manually
- Other

3 Case Study

Case Study

- A controller product with the following:
  - A core code base
  - Custom application components
  - A need to synchronize product and user interface features

Case Study

Version Management

- Separate your technical layers into different repos!
- Utilize one repo for “mission” specific configuration
- Use Git modules to build the specific application

Case Study

User Interface Configuration based on YAMLs

User Interface Configuration based on YAMLs

How do you synchronize your GUI to your embedded system?
- Configuration files
- Interface control documents
- The hope and a prayer method
- Other

4 Going Further

Thank you for attending

Please consider the resources below:

- www.beningo.com
  - Blog, White Papers, Courses
  - Embedded Bytes Newsletter
    - http://bit.ly/1BAHYXm
  - Embedded Software Design
    - https://bit.ly/3PZCtNO

From www.beningo.com under

- Blog > CEC – Embedded Software Design Techniques

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

Digi-Key ELECTRONICS

BENINGO EMBEDDED GROUP

informa markets