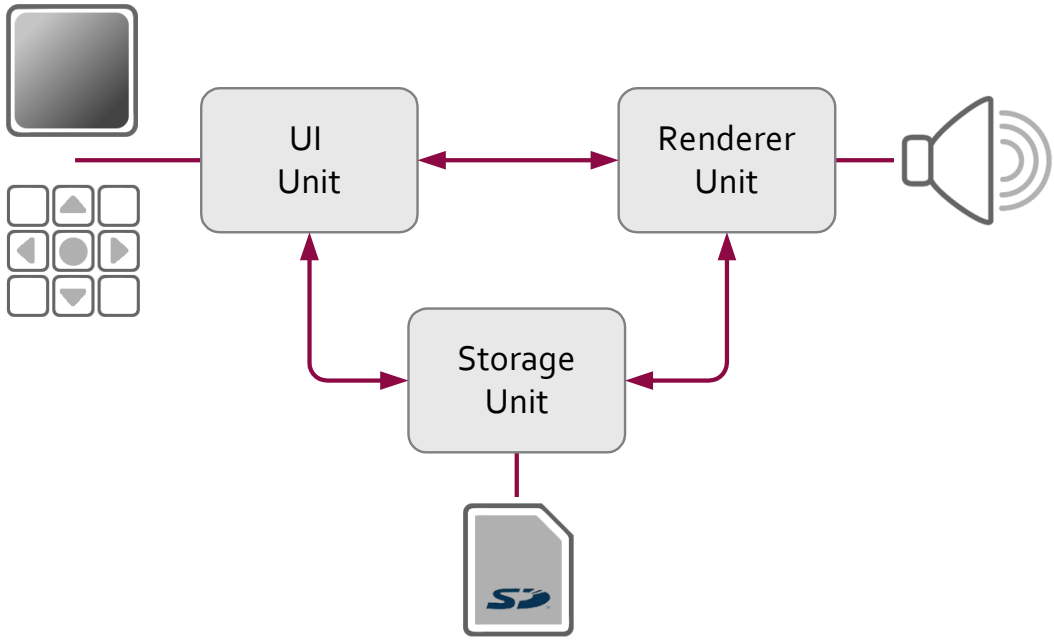
embenatics System Description

This white paper introduces the embenatics tool chain, specially the description of system resources based on a simple MP3 player example.

Introduction to the Sample Application

In this series of white papers a simple MP3 player is used as a sample application to show the various features and working steps of the embenatics tool suite. The figure below shows the building blocks for the MP3 player example. A short description of each block will facilitate the understanding of the basic functionality of the player. Each white paper will focus on different subjects of the sample application design to highlight the specific topic of the paper.



Block Diagram of the MP3 Sample Application

The UI Unit handles the user interaction, which comprises user commands as well as displaying status information, like which track is playing, the list of selected titles, etc. Access to the Storage Unit is used to provide all information about the available music tracks. The UI Unit interfaces with the Renderer Unit to pass on user commands, as well as to display the current status of the player.

© 2011

The Storage Unit keeps track of the available music titles and provides access to the stored MP3 files. It interfaces with the UI Unit to provide track information. The interface with the Renderer Unit provides access to the coded music data that should be replayed.

The Renderer Unit is responsible for handling the music track to be played, managing a play list of titles, displaying information on the state of the player and converting the coded MP3 data into audible music. It receives track information and control commands via the UI Unit interface. Access to the MP3 data is obtained by interfacing with the Storage Unit.

In this white paper the MP3 player application is implemented by using five threads. The DISPLAY thread performs the graphic commands for drawing icons and text, and utilizes the LCD driver for visualization purposes. The KEYPAD thread receives information about key presses from the keypad driver. It forwards the key presses to the UI thread that controls the entire MP3 player. Additionally, there is the STORAGE thread managing a database containing the information about tracks, albums and artists stored on the memory card. A RENDERER thread is responsible for accessing the audio hardware and informing the UI about the audio renderer's state.

Figure 1 illustrates the thread model of the MP3 player in a UML-like notation. Each thread is displayed showing its name and its properties which are limited to thread priority and stack size in this simplified example. Additionally, the diagram shows the interfaces implemented by each thread. The RPC based communication relationship among the threads is indicated by the arrows between the threads, e.g. the UI thread accesses services of the display_control interface of the DISPLAY thread.

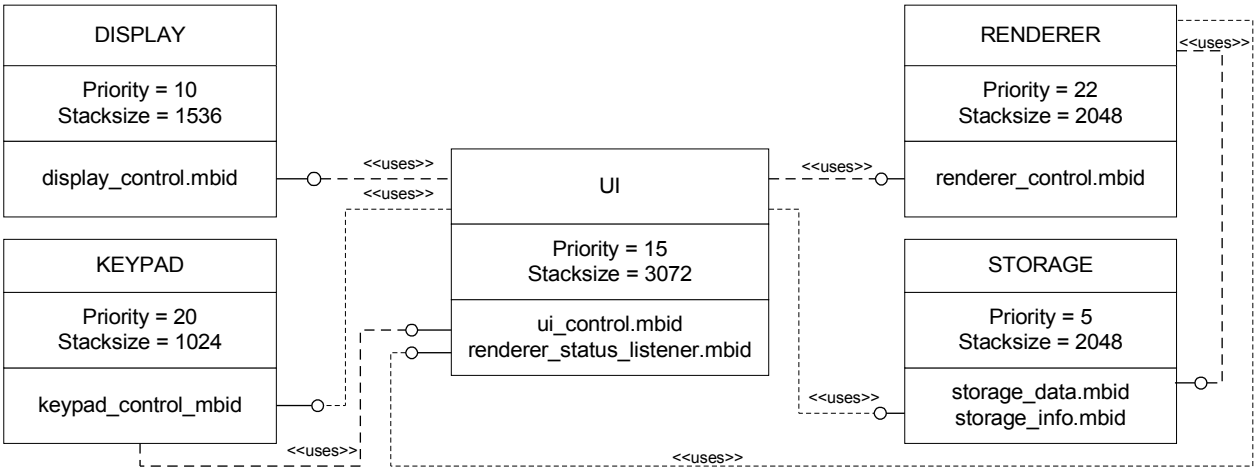


Figure 1 Thread model of the MP3 Player.

The System Description

The embenatics design methodology is based on a centralized definition of system immanent properties like communication interfaces, system resources and the distribution of threads over the subsystems of your application. The description documents created in the system design phase are

the key input during development, diagnosis and testing. The system description document (MBSD) provides the configuration of system resources like threads, memory pools, synchronization objects and their properties as well as the communication relationship among application threads for a project.

The embenatics system description editor mbEdit supports the user in creating the system resource configuration. Each system resource is created and edited using this Eclipse™-based editor, which is shown in Figure 2.

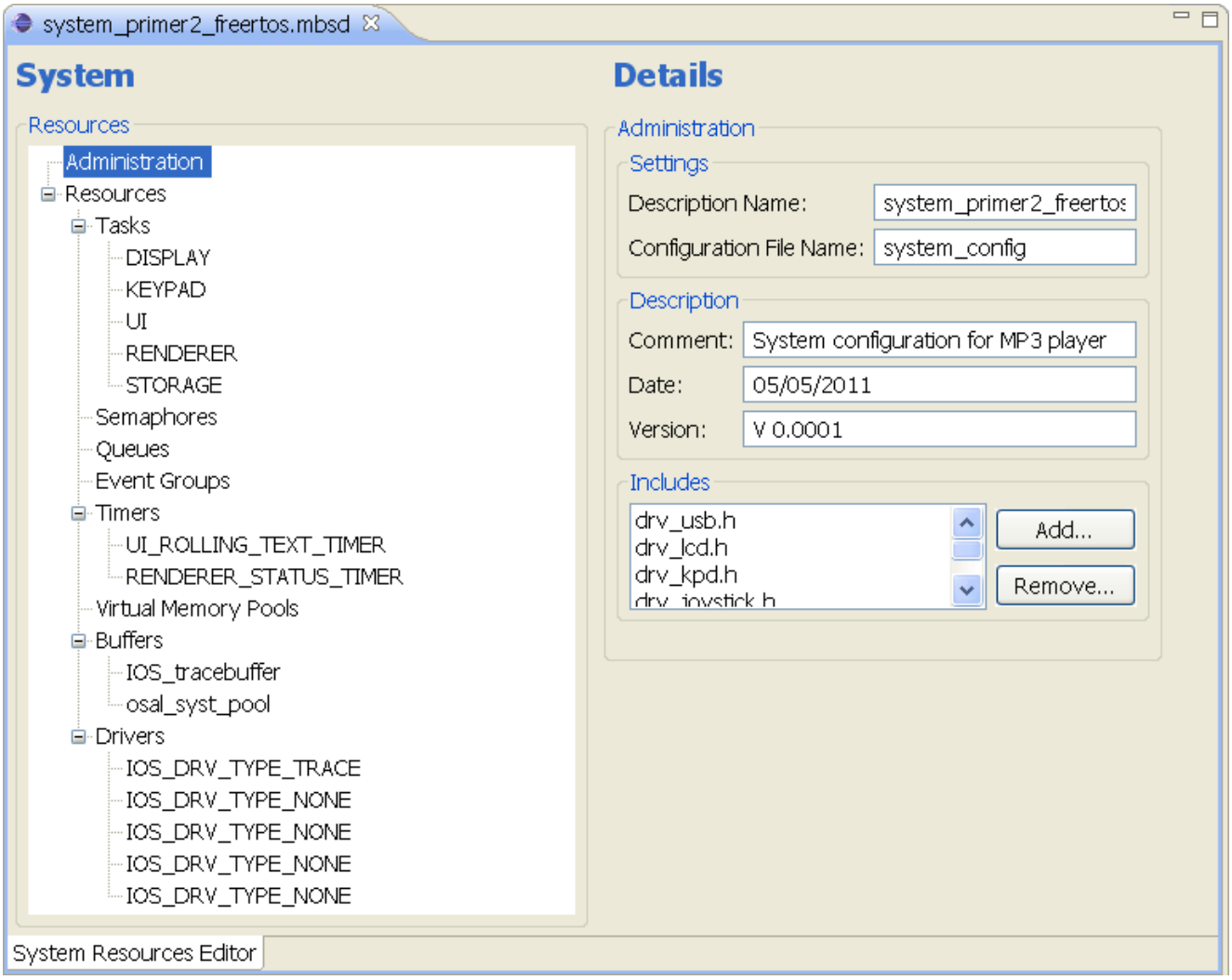


Figure 2 System Resource Editor

A tree on the left displays all OS resources of the application and supports software developers and architects in maintaining and optimizing the system resource configuration. On the right an appropriate form for each type (threads, semaphores, message queues, event groups, timers, virtual memory pools and drivers) is provided to enter the resource attributes. All system resources defined in the system resource description are automatically created by the embenatics foundation layer mbLay during start-up. For further information on the mbLay architecture please see also another document of this series of white papers [mbLay Architecture and Interface].

Administration

The system administration form needs to be filled in before the resource attributes are entered.

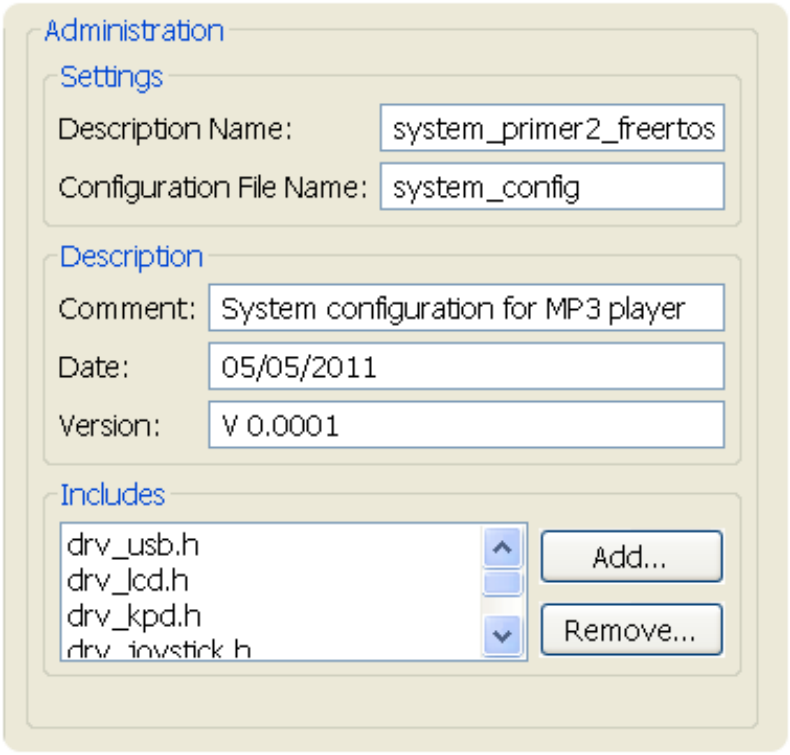


Figure 3 System Administration Form

In the *Settings* section, the name of the system description, which can be different from the file name, must be entered. At the same time, the name of the generated system resource configuration C-file is filled in, which will later be generated by the embenatics generator tool mbGen. The description section is self-explanatory; in the *Includes* section, the header files needed in the system resource description C-file must be entered. Required header files may provide OS dependent priority definitions or the types of the driver configuration data structures.

Threads

When the administration work is done, the system resources need to be configured. The following figure shows the properties of the UI thread. For the MP3 player example we have to add the threads and their properties that were introduced in Figure 1.

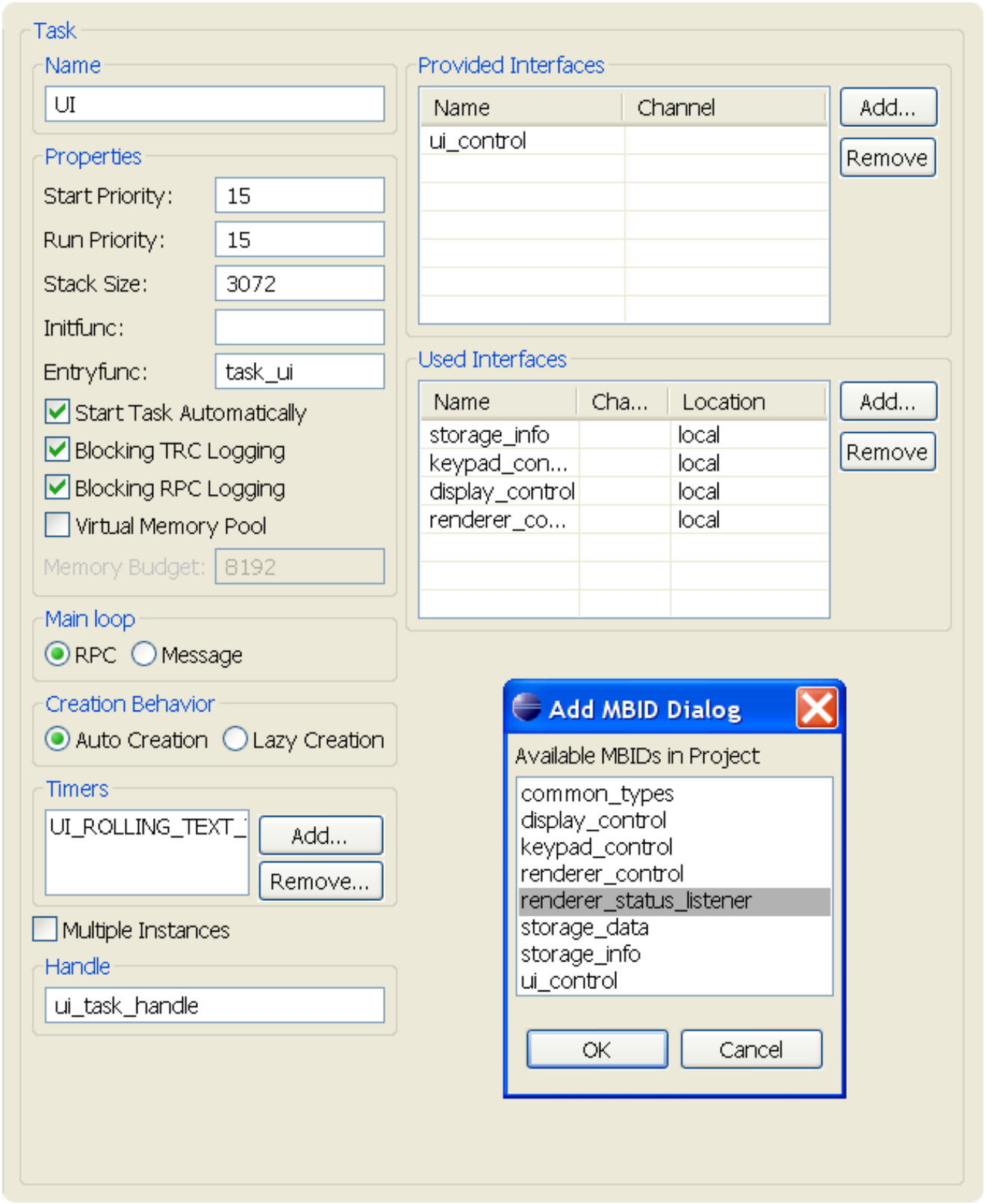


Figure 4 Editor Form for the Thread Properties

In this white paper, we demonstrate the thread configuration for the UI thread. It is the main control thread of the MP3 player. It has to react to key board events detected by the KEYPAD thread and

RENDERER status changes. It also needs to request the required track information from the STORAGE thread to update the DISPLAY content if required. Finally, the UI thread needs to inform the RENDERER as to which tracks will be played.

For the OS-dependent priority we have to distinguish between a so-called *Start-Priority*, which determines the order in which the threads enter their thread function, and the *Run-Priority*, which is set when the start-up is completed. The UI task uses a medium priority of 15 for both, start and run priority. After setting the stack size to 3072 there is the option to provide an init function that is called by mbLay from outside the application thread context before any application thread is created. Initialization of hardware blocks or global data may be performed from within this function. Because the UI task does not need such an init function, the field is left blank. In any case, a thread entry function, here task_ui(), needs to be provided, which is the first function called by mbLay from within the context of this thread.

The following check boxes determine the dynamic behavior of the thread in mbLay. The UI thread will be started automatically. The alternative is to start it manually at a later point in time, by calling the appropriate mbLay API function. The following two check boxes determine the logging behavior. The UI thread should be blocked if the trace buffer is exhausted and has to wait until buffer space becomes available. The alternative is to simply drop the logging event in an out of trace memory condition and continue operating. This setting can be adjusted separately for the debug logs instrumenting the source code and for the built-in logging of the RPC based inter-process communication on a per thread basis.

Finally mbLay provides the option to assign a so-called virtual memory pool to a thread. This mechanism can be used to assign a limited amount of memory to a thread. For further information on virtual memory pools please see another document of this series of whitepapers [mbLay Architecture and Interface]. In this example, the UI thread does not use this feature.

mbLay supports RPC based inter-process communication as well as the exchange of messages. The UI thread of the MP3 player uses RPC and therefore the RPC button in the form is selected. For further information on the communication model please see another document of this series of white papers [mbLay Inter-Process Communication Model]. Nevertheless, mbLay also provides message-based inter thread communication, for which the *Message* button needs to be selected.

Now the OS and mbLay related properties of the thread are completed and the RPC based communication model needs to be configured. The RPC based communication model requires that all interfaces are specified in interface description documents, called MBIDs. For further information on interface description files please see the other document of this series of white papers [embenatics Interface Description]. As we can see in Figure 1, the UI thread implements the interfaces `ui_control` and `renderer_status_listener`. These interfaces are added to the list of the *Provided Interfaces* section in a simple *Add* dialog that allows the selection from a drop-down list of all interfaces which are part of the MP3 player project. In the screenshot shown in Figure 4 the `ui_control` interface is already provided by the UI thread, whereas the `renderer_status_listener` interface is just being added in the dialog displayed in the lower right corner of the screen shot.

The UI thread calls services of the interfaces `renderer_control`, `storage_info`, `keypad_control` and `display_control`. These interfaces are added to the *Used Interfaces* section via the *Add* dialog that allows to select from all interfaces that are implemented by any other thread of the MP3 player project.

Timers

The RENDERER thread uses the RENDERER_STATUS_TIMER to inform all registered listeners periodically about its status changes. The properties of the timer are entered in the timer form. Application timers in mbLay are assigned to threads and the timer will be added to the RENDERER thread via a simple *Add* dialog in the task form.

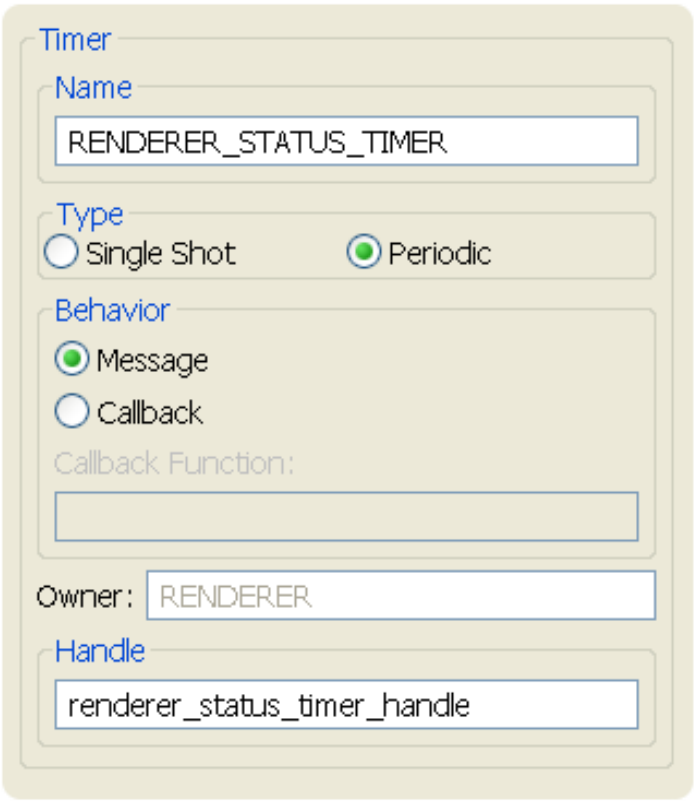


Figure 5 Editor Form for Timer Properties

Besides its name, a timer can be either *Single Shot* or *Periodic*. The renderer status timer of the MP3 player expires periodically and the *Periodic* button is active. In this example, the renderer status timer is a message timer. This means that a timeout message is sent to the renderer thread's message queue when the timer expires. In this case, the timeout is processed in the context of the owning thread. The alternative would be to configure the timer as a *Callback* timer. In that case, a time-out function provided by the owning thread is called from outside the thread context.

Memory Buffers

In the *Buffers* section at least two memory buffers need to be present. The `osal_syst_pool` is used by mbLay for the allocation of the resource control blocks and thread stacks. The IOS_tracebuffer is used to store the logging events. In the MP3 player example this buffer has a size of 8192.

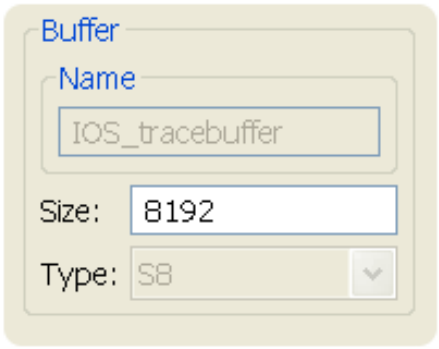


Figure 6 Editor Form for Memory Configuration

Drivers

In addition to the system resources, the drivers of the MP3 player need to be configured. All drivers are implemented by using the embenatics driver architecture called IOS. The drivers configured in the system resource editor will be initialized by mbLay. The configuration of the logging interface driver is shown in Figure 7; it works in a similar way for the display, keypad, file system and audio driver.

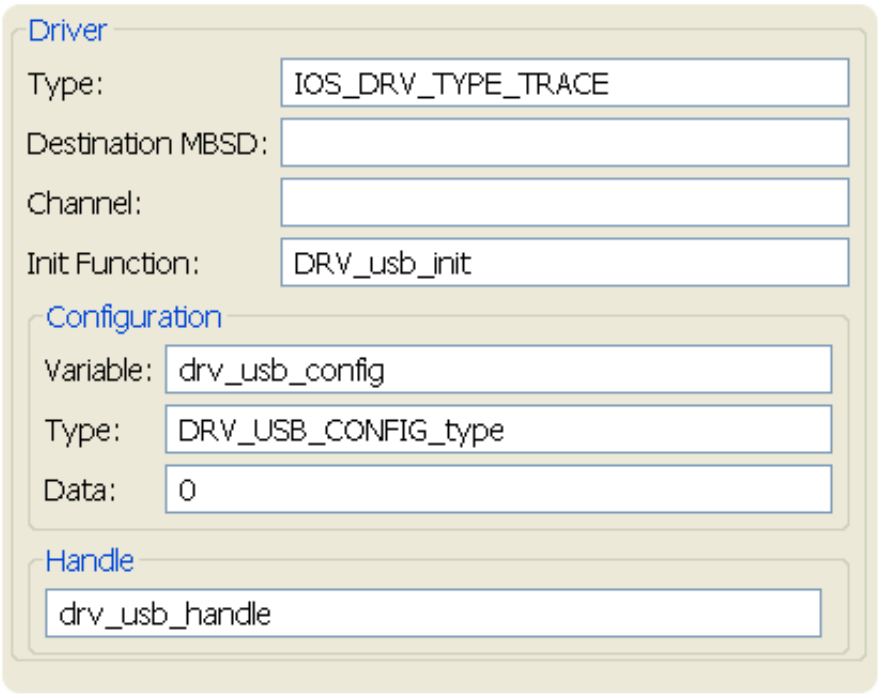


Figure 7 Editor Form for Driver Configuration

First, the driver type needs to be selected, which is IOS_DRV_TYPE_TRACE for the logging driver. For the other drivers use IOS_DRV_TYPE_NONE. The fields *Destination MBSD* and *Channel* are not required in the single-core MP3 Player example - they are required in the multi-core case. For further information on embenatics multi-core support see the other document of this series of white papers [Designing Multi-Core Applications].

The driver's initialization function is entered in the following field. The MP3 player uses a USB driver for the logging interface; the respective name of the initialization function `DRV_usb_init` is entered. For the driver configuration, the type of the driver configuration structure needs to be specified followed by the configuration data itself. This allows the generator tool mbGen to generate a C-structure containing the driver configuration data. Finally, a driver handle needs to be provided that will be used for the driver access.

Processing the System Description Document

When the resource configuration is complete, the embenatics generator tool mbGen is called. It processes both, this system description document and the interface description documents of this project, and generates a set of output files which will be used during the development and diagnostic phase. The tables generated from the system description document will be used for the automated boot-up as well as for the RPC based inter-process communication routing. For further information on mbGen see the other document of this series of white papers [embenatics Design Methodology].

About Us

embenatics is a new company that entered the market in 2010. Our focus is on embedded software development; as such we offer a software foundation layer and tool suite that supports your development team in designing embedded software in an efficient, portable and maintainable way. Based on our wide and varied experience in embedded systems design and development, we know that future product requirements are hard to predict. Our goal is, therefore, to provide you with our technology to make the design of your products as flexible and adaptable as possible. Our approach allows your company to concentrate on the core competencies that differentiate your valuable product from those of your competitors.

Before embenatics was founded, we worked with well-known international companies over two decades and gained valuable experience in the embedded software business. While working as software developers and architects, we encountered the various challenges of the embedded software development life cycle. This wide range of experiences is the backbone of the software foundation products that are offered by embenatics.

Our business philosophy is to establish a close and trustful relationship with our customers in order to successfully promote and support projects over a long time period. For further information please contact

Joachim Pilz
Beerenstraße 29
14163 Berlin

info@embenatics.com
www.embenatics.com

Phone +49 30 26 34 75 28
Mobile +49 176 96 98 46 07