# Adding a Real-Time task model

## Purpose

This document details an improvement of the DSX task modeling API in order to add real-time capabilities.

## Real-Time Task features

Real-time tasks have:

- No implicit permanent existance (they are not repeated infinitely)
- Possibility to receive asynchronous signals from:
    - Other Real-time tasks
    - "normal" tasks
    - External interrupt lines
    - Programmable timers
- An user-defined set of slots (receivers for signals)
- one "main" behaviour function
- an internal state

## Restrictions

Having code called asynchronously from slots implies the task's slot could interrupt "main" behaviour function. Therefore:

- slots are executed in a separate limited stack space, no heavy computation should be done in them.
- as main behaviour function may be already communicating on ports, the only permitted operations on ports are reset, see below.
- if any port is touched (reset) from a slot, task must be resetted too (see `srl_task_*()` below).

## Description API Evolution

Let's introduce three new objects in DSX description language:

- SignalPort,
- SlotPort,
- RtTaskModel
- Signal

### New Port Types

#### Signal

A SignalPort port is an **output** for an event propagation. It has no parameter:

```
...
ports = {...
         'finished_processing' : SignalPort(),
         },
...
```

### Slot

A SlotPort is an **input** for an event propagation. It has no parameter:

```
...
ports = {...
         'on_finish_processing' : SlotPort(),
         },
...
```

## Enhancement of the current TaskModel

It is now possible to declare `SignalPorts` for a TakModel.

## New RtTaskModel

It looks like a TaskModel object, but its ports may contain Slot and Signal ports:

```
RtTaskModel("processing_monitor",
            ports = {
                     'on_begin_processing': SlotPort(),
                     'on_finish_processing': SlotPort(),
                     'on_application_reset': SlotPort(),
                     },
```

Implementations are listed as usual. As it requires a good amount of supporting code, for now, the only valid implementation for a Real-time task will be software.

```
            impls = [
                SwTask('main_behaviour',
                       stack_size = 4096,
                       bootstrap = 'optional_bootstrap_func',
                       sources = ['path/to/source.c'],
```

Now we have to list the Real-time tasks specific attributes:

- The internal state data C type
- The slots functions available
- An optional mapping of some slots to slots functions. By default, mapping is done by a name identity (here `on_application_reset` will implicitely be routed to the `on_application_reset` function), but this permits to route many slots to one function.

```
                       state_type = 'struct process_monitor_state_s',
                       slot_funcs = ['something_happened', 'on_application_reset'],
                       slot_map = {'on_begin_processing': 'something_happened',
                                   'on_finish_processing': 'something_happened',
                                   },
                    ),
                ],
            )
```

## Tcg evolution

Now the Tcg has to connect signals to slots.

Connection is done through a `Signal` object, assigned during the portmap.

The following restrictions must be respected:

- A signal must be connected to at least one `SignalPort`
- A signal may be connected to at least zero `SlotPort`
- A `SlotPort` may be left unconnected, and may be connected to one `Signal`
- A `SignalPort` must be connected to exactly one `Signal`

### Mapping

Mapping of `Signal` resources on `MutekS` or `MutekH` needs:

- a `desc` in (possibly) read-only memory

## SRL API evolution

### SignalPort-related

There is a new abstract type: `srl_signal_t`. The only thing permitted on a SignalPort is to emit the signal, therefore, the only function available for it is `srl_signal_emit()`.

```
FUNC(processor)
{
    srl_signal_t finished_processing = GET_ARG(finished_processing);


    ...

    srl_signal_emit(finished_processing);
}
```

### SlotPort-related

There is nothing publicly available to deal with slots from the software point of view, SlotPorts are not even exposed to GET_ARG().

### Task Management Related

There are now ways to tell the "main" function to start or stop. As this function is the only one, it is implicit in the API. It can't be started more than once. The two available functions are:

- `srl_task_launch();`
- `srl_task_terminate();`

If the task is already running, `srl_task_launch()` has no effect. If the task is not currently is running, `srl_task_terminate()` has no effect.

Terminating a task means it will be interrupted whatever it is doing, even if it has taken a lock or is operating on a communication channel. That means you **MUST** reset **all** resources (ports) when resetting a task.

In order to restart a task, simply `terminate()` and `lauch()` it.

## Reset operations for existing communication channels

We introduce the following new calls:

- `srl_mwmr_reset( srl_mwmr_t channel )`
- `srl_lock_reset( srl_lock_t lock )`
- `srl_barrier_reset( srl_barrier_t barrier )`

Those calls don't abide current locking state of the resources and resets their locks to an "free" status. Concurrent tasks using the same resources may reach an undefined state. These tasks should be reset too.