BTF-Specification

Version History

Version	Author	Datum	Description
V1.0	[Timing-Architects]	2011-07-18	Initial specification approved with thanks by Continental Automotive GmbH, extended by source-entity-instance column
V2.0	[Timing-Architects]	2012-04-17	Added new data types
V2.0.1	[Timing-Architects]	2013-03-29	Added state charts and description of all entities
V2.0.2	[Timing-Architects]	2013-04-22	First public release
V2.1.0	[Timing-Architects] [Robert Bosch GmbH]	2013-06-18	- Changed Process State Chart for compliance to OSEK 2.2.3 Extended Task Model - Some improvements of description
V2.1.1	[Timing-Architects]	2013-10-30	Clarified description and examples regarding difference preempt/suspend for processes/runnables.
V2.1.3	[Timing-Architects]	2014-04-10	Process state chart: changed layout according to OSEK state order. First published version.

Note: In version key V x.x.y, x represents change in BTF, y is only specification update.

This specification was extended in the context of the AMALTHEA ITEA2 project (ID 01IS11020E) and is released on the Eclipse Auto IWG.

CONTENT

1. LIST OF FIGURES

2. LIST OF TABLES

1. INTRODUCTION

This document specifies a tracing format for timing evaluation of event based systems. The BTF (Best Trace Format, originating from Better Trace Format (BTF V1.0)) is a CSV-based format for representation of event-traces in ASCII. BTF is a format definition for full scale timing traces of simulator and profiling tools.

The Best Trace Format (BTF) is based on the Better Trace Format, initially defined by Continental Automotive GmbH. It allows analyzing the behavior of a system in a chronologically correct manner in order to apply timing, performance, or reliability evaluations. In general, it assumes a signal processing system, where one component of the system notifies another component of the system. These notifications are realized by events, stored in the BTF file. In comparison with compact trace formats from debugger traces, a BTF log of an event includes the entire information, namely: which component interacts with which component by an event.
Advanced scheduling concepts may be used in multicore processor systems where one traced component may have multiple instances at the same time, i.e. global scheduling or task migration concepts. This requires instance identification in order to derive which instance of a component is addressed in the event log. This means for example that each task execution can be exactly identified for the complete lifetime from activation till termination by its component name and instance counter.

The following figure shows the interaction between two component instances, where component Name1 (Instance #21452) sends an event X to component Name2 (Instance #124) at t=1200025. A component instance is generated from its parent component and duplicates its behavior (e.g. execution time according to a certain sequence). Nevertheless, it may be possible that a component instance exists over the complete traced time interval.

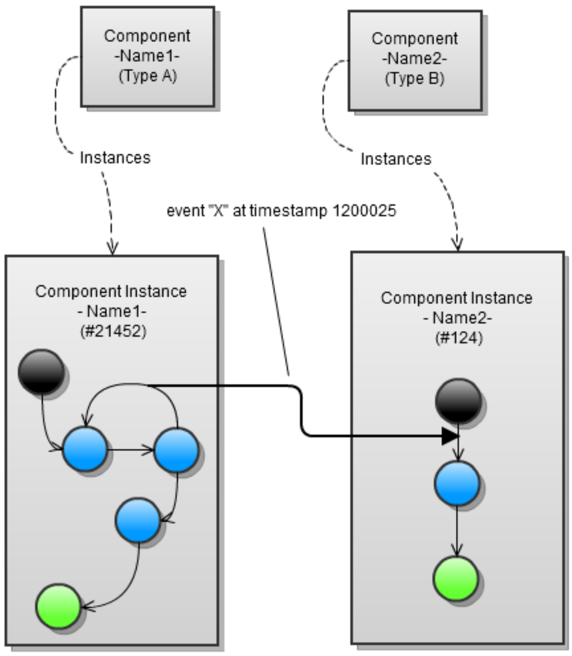

FIGURE 1: SCHEMATIC VISUALIZATION OF INTERACTION BETWEEN TWO ENTITY INSTANCES.

2. Structure of BTF-File

A BTF-file consists of two parts:

1. A header-section, containing meta-information on objects of the trace and optional comments.

   All lines start with a ‚#'. The meta-information is described by pragma-statements (see Section 2.1.2). Comments contain directly after the '#'-symbol a space, which allows the differentiation from pragma-statements.

2. A data-section, containing the trace-data of the simulation or measurement.

   The data-section consists of lines in CSV format with optional comment-lines. Each line represents one event of the traced system. The columns of the event-line describe the time, entities, and event. Comments are defined as in the header-section.

For the representation of the data-sections two ways exist. The symbolic-mode describes entities and event by names. The numeric-mode describes entities and event by a numerical identifier. In this case, the header-section includes the mapping between numerical identifier and a string of the name.

2.1. Header

The header includes parameters, used for the interpretation of the trace or information of the trace generator, and comments. Parameters and comments are indicated by a '#'-symbol.
The typical header of a BTF-file includes at least the version, creator, creation date and the time scale. Further information is optional.

2.1.1. Comments

Each row, starting with a '#'-symbol which is followed with a whitespace is a parameter. Comments can be part of the header or can be entered at any position of the data section.

2.1.2. Parameters

Each row, starting with a '#'-symbol and one of the following parameter definitions is a parameter.
The parameter definition may not start with a whitespace. When the symbol '-' follows the '#' symbol, the row is an entry of the last defined table (e.g. typeTable). Following parameter definitions are predefined:

TABLE 1: PARAMETERS FOR BTF HEADER SECTION

Parameter	Description	Type	Example
#version	Version of BTF format definition	String	#version 1.0
#creator	Name and version of the program or device, which generated the trace	String	#creator TA-Simulator (12.10.2.47)
#creationDate	Timestamp of the start of simulation or measurement. The format has to comply with "ISO 8601 extended specification for representations of dates	String (ISO 8601)	#creationdate 2012-09-02T16:40:30Z

This specification was extended in the context of the AMALTHEA ITEA2 project (ID 01IS11020E) and is released on the Eclipse Auto IWG.

	and times" YYYY-MM-DDTHH:MM:SS. The time should be in UTC time (indicated by a “Z” at the end)		
#inputFile	Filename of the model which was used for the simulation	String (URI)	#inputFile D:\Workspace\Project\DualCore.rte
#timescale	Defines the resolution of the timestamps in the trace. Default unit is nano-seconds (ns).	String (Enumeration [ps,ns,us,ms,s])	#timescale ns
#typeTable	Indicates the beginning of a mapping from all entities to a numerical Type-Id. See Table 3 for existing types. Type-Ids start with 0. Missing Ids are allowed	-<n> String	#entityType #-0 T #-1 R #-2 SIG
#entityTable <n>	Indicates the beginning of a mapping from all entities to a numerical Entity-Id. An entity can be a task, runnable, etc. Type-Ids start with 0 and some Ids can be missing.	-<n> String	#entityTable #-0 Task_1ms #-1 GetSignal #-2 Main #-3 Temperature
#entityTypeTable <n>	Indicates the beginning of a mapping from all entities to types. Both, entity and type has to be defined before in the entityTable and entityTable.	-<n> String	#entityTypeTable #-T Task_1ms #-R GetSignal #-R Main #-SIG Temperature

Example:

A typical header can look in the following way:

```
#version 1.0
#creator TA-Toolsuite 12.06.1
# Simulation of dualcore processor 120MHz, 16Kbyte RAM
#creationDate 2012-08-31T15:53:00
#inputFile c:\TAsc\doc\examples\ems.tap
#timeScale ns
#entityType
#-0 T
#-1 R
#entityTable
#-0 Task_1ms
#-1 Task_2ms
#-2 Runnable_1ms_Init
#-3 Runnable_2ms_Store
#-4 Runnable_2ms_Read
#entityTypeTable
#-T Task_1ms
#-T Task_2ms
#-R Runnable_1ms_Init
#-R Runnable_2ms_Store
#-R Runnable_2ms_Read
```

2.2. Data Section

The trace information is represented in CSV format. Each line describes one event. The interpretation of one line depends on the event type, shown in the next section.

For separating the content of one line the symbol ',' (comma) is used. For the case using floating numbers, a '.' (point) has to be used as decimal separator. Strings can be written between quotation marks '"'. For strings with space characters writing in quotation marks is forced.

At any point of the trace section, a comment with pragma '#' can be written.

2.3. Entities and Events

The data section consists of line by line interpreted data. Each line has eight columns, whereas the last column is optional. A line contains the following elements:

<Time>,<Source>,<SourceInstance >,<TargetType>,<Target>,<TargetInstance>,<Event>,<Note>

The interpretation of the different columns depends on the <TargetType>-column.

TABLE 2: DESCRIPTION OF BTF COLUMNS

Column	Name	Description	Relevant for entity type
1	Time	Integer timestamp for one action. The timescale is given in the configuration section by the parameter #timescale.	all
2	Source	Unique name for the source which triggers the event. (e.g. Core at start of a task or stimulus at the activation of a task)	all
3	SourceInstance	Instance counter for the source. Non-instanceable entities (e.g. core) have each time the instance '-1'. Instanceable entities like stimuli starting with 0 and increment at each instantiation the counter. If there is no information for the instance this field can be empty.	all
4	Type	Type of the event target.	all
5	Target	Unique name for the target, which triggers the event (e.g. a task, runnable, signal-access).	all
6	TargetInstance	Instance counter for the target.	all
7	Event	Name of the event	all
8	Note	Optional field for further information (e.g. signal value at signal read or write access)	SIG, SEM

The fourth column (<TargetType>) includes the type of the event. Following types are defined in the BTF:

TABLE 3: ENTITY TYPES.

Category	Type-ID	Name	Description
Environment			
	STI	Stimulus	Trigger point for a Task or ISR
Software			
	T	Task (Specialization of Process)	Object handled by OS Scheduler, and calling all "Top-Level" Runnables. A Task is the specialization of a Process type.
	ISR	Interrupt-Service-Routine (Specialization of Process)	Object handled by Interrupt-Management Unit and calling all "Top-Level" Runnables. An ISR is the specialization of a Process type.
	R	Runnable	Object, called by a Process or another Runnable.
	IB	Instruction block	Sub-fraction of a Runnable
Hardware			
	ECU	Electronic Control Unit	Hardware device which has at least one processor.
	P	Processor	Hardware device which has at least one core
	C	Core	Hardware device which is part of a processor and executes software.
Operating System			
	SCHED	Scheduler	Part of operating system which assigns processes to cores.
	SIG	Signal	Shared data object (e.g. variable) in a software.
	SEM	Semaphore	Operating system object, for restricting access to resources.
Information			
	SIM	Simulation	Used for notification events from simulation environment, e.g. simulation started or simulation stopped.

Example:

```
0,        Task_A,   0,   T,   Task_A,          0,   activate
100,      Core_1,   0,   T,   Task_A,          0,   start
100,      Task_A,   0,   R,   Runnable_A_1,    0,   start
6766,     Task_A,   0,   R,   Runnable_A_1,    0,   terminate
6766,     Task_A,   0,   R,   Runnable_A_2,    0,   start
10000,    Task_B,   0,   T,   Task_B,          0,   activate
10100,    Task_A,   0,   R,   Runnable_A_2,    0,   suspend
10100,    Core_1,   0,   T,   Task_A,          0,   preempt
10100,    Core_1,   0,   T,   Task_B,          0,   start
10100,    Task_B,   0,   R,   Runnable_B_1,    0,   start
16766,    Task_B,   0,   R,   Runnable_B_1,    0,   terminate
16766,    Core_1,   0,   T,   Task_B,          0,   terminate
16866,    Core_1,   0,   T,   Task_A,          0,   resume
16866,    Task_A,   0,   R,   Runnable_A_2,    0,   resume
20199,    Task_A,   0,   R,   Runnable_A_2,    0,   terminate
20199,    Core_1,   0,   T,   Task_A,          0,   terminate
```

Gantt-Chart of Example

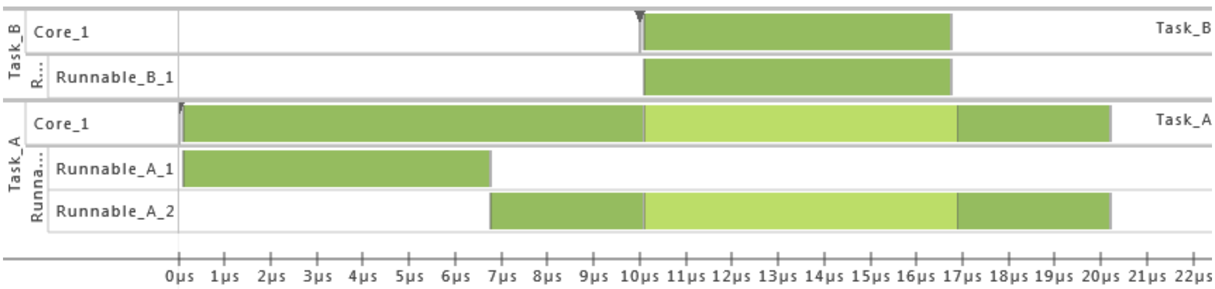

FIGURE 2: GANTT CHART OF EXAMPLE. DARK GREEN AREAS SHOW EXECUTION OF TASK OR RUNNABLE. LIGHT GREEN AREAS SHOW TASKS/RUNNABLES IN PREEMPT/SUSPENDED STATE.

2.3.1. STIMULUS-EVENTS

A stimulus is used to model external inputs or internal behavior, which is not modeled by other software or hardware parts. A stimulus is able to activate a task/ISR or set a signal value.

TABLE 4: COLUMNS FOR STIMULUS ENTITY.

Column	Entries
<Source>	Simulation (SIM), Task (T) or ISR (ISR)
<Event>	trigger

Example:

1200150,	TASK_InputProcessing,	0,	STI,	IPA_InputReady,	0,	trigger
1200237,	SIM,	-1,	STI,	IR_SCHED_Tasks_C1,	7,	trigger
1222581,	SIM,	-1,	STI,	IR_SCHED_Tasks_C1,	8,	trigger
2239187,	TASK_InputProcessing,	1,	STI,	IPA_InputReady,	1,	trigger
2250000,	SIM,	-1,	STI,	TIMER_1ms,	2,	trigger

This specification was extended in the context of the AMALTHEA ITEA2 project (ID 01IS11020E) and is released on the Eclipse Auto IWG.

2.3.2. Process-Events (Task- and ISR-Events)

A process can be either a task or an interrupt service routine. A process is activated by a stimulus, as described in section 2.3.1. After activation, a scheduler assigns the process to a core where the process is executed. A running process can be preempted by another process and change to READY. Alternatively, a cooperative process can change itself to READY, e.g. at a schedule point or explicit migration to another core. When a running process requests a resource (e.g. semaphore or event) which is not available, the process waits actively (e.g. "while(ResourceNotAvailible){…}"). This is indicated by the state POLLING. The scheduler could decide to remove a waiting process from the core and the process changes in state PARKING (passive waiting). When the requested resource becomes available but the process is in state parking, the process changes again to state READY.

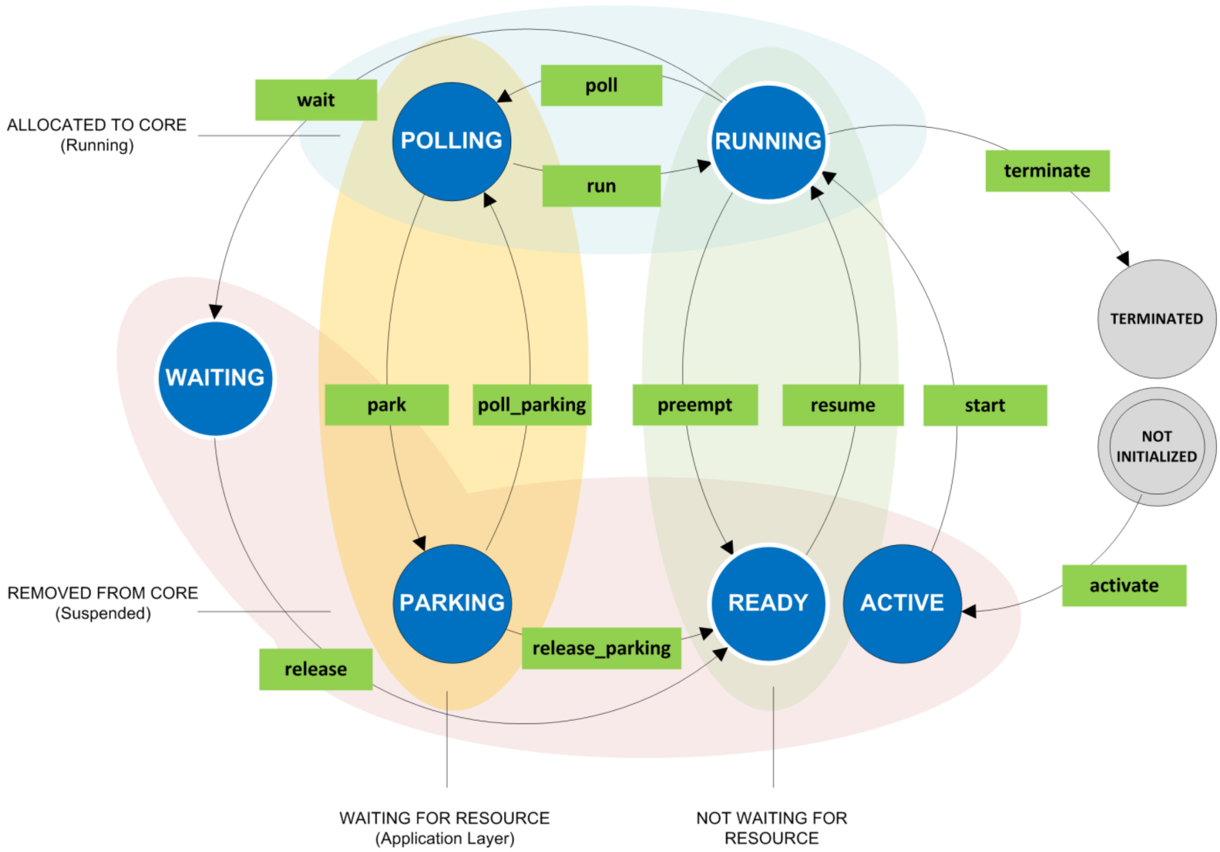

Figure 3: Process State Chart.

***Note**: Whenever process (P) is used in the following description, this can either be a task (T) or interrupt-service-routine (ISR)*

Table 5: Columns for process entity.

Column	Entries
<Source>	Stimulus (STI), Core (C), process (P)
<Event>	activate, start, preempt, resume, terminate, poll, run, park, poll_parking, release_parking, wait, release, deadline, mpalimitexceeded, boundedmigration, phasemigration, fullmigration, enforcedmigration

TABLE 6: STATES FOR PROCESS ENTITY.

State	Description
ACTIVE	When instance is ready for execution
RUNNING	When instance executes on a core
READY	When instance was preempted
WAITING	When instance has requested an OS Event which is not available and waits passively
POLLING	When instance has requested a resource which is not available and waits actively
PARKING	When instance waits for a not available resource and is preempted
TERMINATED	When instance was finished execution

TABLE 7: EVENTS FOR PROCESS ENTITY.

Internal Event	Description	Source
ACTIVATE	Process instance is activated by a stimulus	STI, P
START	Process instance is allocated to the core and starts execution for the first time	C
PREEMPT	Executing process instance is stopped by the scheduler, e.g. because of a higher priority process which is activated.	C
RESUME	Preempted process instance continues execution on the same or other core.	C
TERMINATE	Process instance has finished execution	C
POLL	Process instance has requested a resource by polling (active waiting) which is not available	C
RUN	Process instance resumes execution after polling (i.e. active waiting) for a resource	C
PARK	Active waiting process instance is preempted by another process.	C
POLL_PARKING	Parking process instance is allocated to the core and again polls (i.e. actively waits) for a resource.	C
RELEASE_PARKING	Resource which is requested by parking process instance becomes available, but parking process stays preempted and changes to READY state.	C (last Core)
WAIT	Process has requested a non-set OS EVENT (see OSEK 2.2.3 Extended Task Model, WAIT_Event()).	C (last Core)
RELEASE	OS EVENT which was requested by a process is set (see OSEK 2.2.3 Extended Task Model, SET_Event()) and process is ready to proceed execution.	C (last Core)

TABLE 8: INFO EVENTS FOR PROCESS ENTITY.

Notification-Event	Description
MPALIMITEXCEEDED	When there are more process instances of this process as the MPA-LIMIT value (MPA = MultipleProcessActivation)
BOUNDEDMIGRATION	When last executing core of previous instance is not equal to first executing core of this instance.
PHASEMIGRATION	When the executing core before a preemption is not equal to the new executing core and there is no schedule point right before this execution.
FULLMIGRATION	When the executing core before a preemption is not equal to new executing core and there is a schedule point right before this execution.
ENFORCEDMIGRATION	When a process migrates at a predefined position in execution to another scheduler.

Example:

The example shows the activation of TASK_InputProcessing, triggered by a timer. TASK_InputProcessing starts execution and is preempted by task TASK_1MS, also triggered by a timer. After TASK_1MS has finished execution, TASK_InputProcessing resumes execution.

```
6150000,      TIMER-A_2ms,   3,     T,     TASK_InputProcessing, 3,     activate
6150100,      Core_1,        0,     T,     TASK_InputProcessing, 3,     start
6250000,      TIMER-1MS,     6,     T,     TASK_1MS,             6,     activate
6250100,      TASK_1MS,      6,     STI,   IR_SCHED_Tasks_C1,    24,    trigger
6250100,      Core_1,        0,     T,     TASK_InputProcessing, 3,     preempt
6250100,      Core_1,        0,     T,     TASK_1MS,             6,     start
6721825,      Core_1,        0,     T,     TASK_1MS,             6,     terminate
6721925,      Core_1,        0,     T,     TASK_InputProcessing, 3,     resume
7110175,      Core_1,        0,     T,     TASK_InputProcessing, 3,     terminate
```

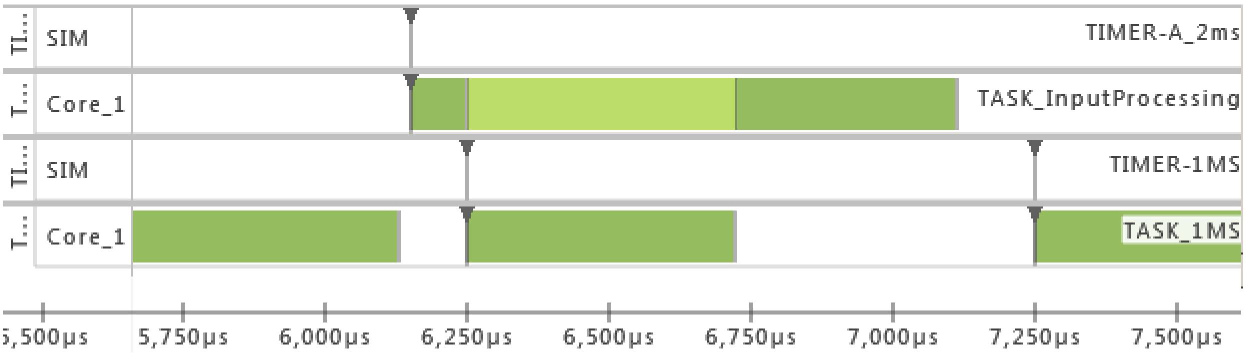

FIGURE 4: GANTT CHART OF EXAMPLE. DARK GREEN AREAS SHOW EXECUTION OF TASK. LIGHT GREEN AREAS SHOW TASKS IN PREEMPT STATE.

2.3.3. Runnable Events

A runnable is called within a process instance or in the context of another runnable. When a runnable is called, it starts and changes to RUNNING. When the process instance which includes the runnable is suspended, the runnable itself is also suspended and changes to state SUSPENDED. When the process instance is resumed, the runnable also changes to RUNNING. After complete execution, the runnable changes to TERMINATED.

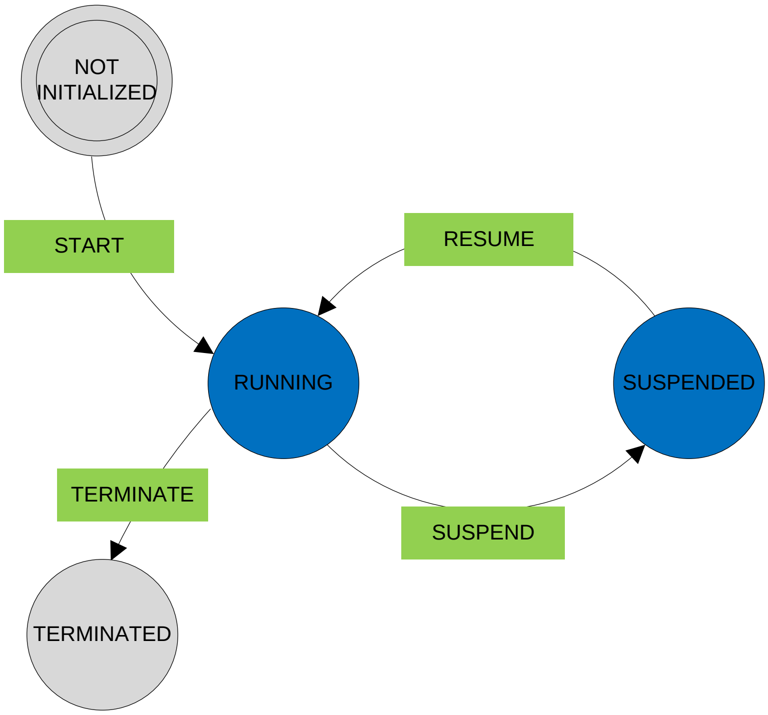

Figure 5: Runnable state chart.

Table 9: Columns for runnable entity.

Column	Entries
<Source>	Process (P)
<Event>	start, suspend, resume, terminate

Table 10: States for runnable entity.

State	Description
RUNNING	Runnable instance executes on a core
SUSPENDED	Runnable instances has stopped execution on a core

TABLE 11: EVENTS FOR RUNNABLE ENTITY.

Events	Description	Source
START	Runnable instance is allocated to the core and starts execution for the first time.	P
SUSPEND	Executing runnable instance is stopped, because the calling process is suspended.	P
RESUME	Suspended runnable instance continues execution on the same or other core.	P
TERMINATE	Runnable instance has finished execution.	P

The runnable R_T20_O2 is started and preempted by runnable R_T1_Init. After termination of R_T1_Init, runnable R_T20_O2 resumes execution.

```
3232706,    TASK_20MS,   0,    R,    R_T20_O2,    0,    start,
3250100,    TASK_20MS,   0,    R,    R_T20_O2,    0,    suspend,
3250100,    TASK_1MS,    3,    R,    R_T1_Init,   3,    start,
3326368,    TASK_1MS,    3,    R,    R_T1_Init,   3,    terminate,
3714350,    TASK_20MS,   0,    R,    R_T20_O2,    0,    resume,
3740812,    TASK_20MS,   0,    R,    R_T20_O2,    0,    terminate,
```

2.3.4. Signal-Events

A signal is a label, which can be accessed by a process instance.

TABLE 12: COLUMNS FOR SIGNAL ENTITY.

Column	Entries
<Source>	Process (P), Stimulus (STI)
<Event>	read, write

TABLE 13: EVENTS FOR SIGNAL ENTITY.

Events	Description	Source
READ	Signal is read by a process	P
WRITE	Signal is written by a process or a stimulus	P, STI

Example:

```
1222481,    STI_MODE_SWITCH,      0,    SIG,    HighPowerMode,      0,    write,  1
1222481,    TASK_200MS,           0,    SIG,    HighPowerMode,      0,    read,   1
4482566,    TASK_WritingActuator, 2,    SIG,    S16_C1_1,           0,    write,  0
5590428,    TASK_10MS,            0,    SIG,    S16_C1_1,           0,    read,   0
```