SART Developer's Guide

1.00

Semantic Aware Real-Time scheduler

September 2010

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

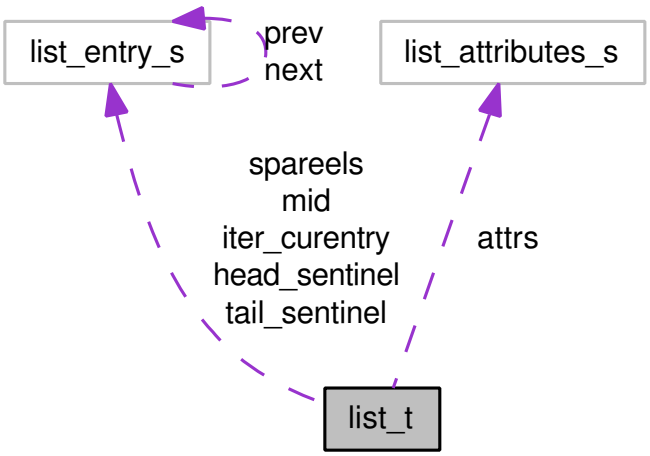
Chapter 3

Data Structure Documentation

3.1 list_t Struct Reference

`#include <simclist.h>`

Collaboration diagram for list_t:



Data Fields

- struct list_entry_s * **head_sentinel**
- struct list_entry_s * **tail_sentinel**
- struct list_entry_s * **mid**
- unsigned int **numels**
- struct list_entry_s ** **spareels**
- unsigned int **spareelsnum**
- int **iter_active**
- unsigned int **iter_pos**
- struct list_entry_s * **iter_curentry**
- struct list_attributes_s **attrs**

3.1.1 Detailed Description

list object

The documentation for this struct was generated from the following file:

- simclist.h

3.2 SRT_CONJUNCTION Struct Reference

`#include <srttypes.h>`

Data Fields

- srt_id **id**

3.2.1 Detailed Description

SART Conjunction

The documentation for this struct was generated from the following file:

- srttypes.h

3.3 SRT_DATA Struct Reference

`#include <srttypes.h>`

Data Fields

- srt_id **id**

3.3.1 Detailed Description

SART Data

The documentation for this struct was generated from the following file:

- srttypes.h

3.4 SRT_DISJUNCTION Struct Reference

`#include <srttypes.h>`

Data Fields

- srt_id **id**

3.4.1 Detailed Description

SART Disjuntion

The documentation for this struct was generated from the following file:

- srttypes.h

3.5 SRT_JOB Struct Reference

`#include <srttypes.h>`

Data Fields

- srt_id **id**

3.5.1 Detailed Description

SART Job

The documentation for this struct was generated from the following file:

- srttypes.h

3.6 srt_job_attr Struct Reference

`#include <sart.h>`

Data Fields

- bool disabled
- u_int prominence
- SRT_POLICY policy

3.6.1 Detailed Description

Job attributes structure

3.6.2 Field Documentation

3.6.2.1 bool srt_job_attr::disabled

Disabled Job

3.6.2.2 SRT_POLICY srt_job_attr::policy

Reasoning policy assigned to the Job

3.6.2.3 u_int srt_job_attr::prominence

Prominence of the Job

The documentation for this struct was generated from the following file:

- sart.h

3.7 SRT_RESOURCE Struct Reference

`#include <srttypes.h>`

Data Fields

- srt_id **id**

3.7.1 Detailed Description

SART Resource

The documentation for this struct was generated from the following file:

- srttypes.h

3.8 SRT_TASK Struct Reference

`#include <srttypes.h>`

Data Fields

- srt_id **id**
- void * **descriptor**

3.8.1 Detailed Description

SART Task

The documentation for this struct was generated from the following file:

- srttypes.h

3.9 srt_task_attr Struct Reference

`#include <sart.h>`

Data Fields

- bool constraint_hard
- bool constraint_forced
- bool constraint_disabled

3.9.1 Detailed Description

Task attributes structure

3.9.2 Field Documentation

3.9.2.1 bool srt_task_attr::constraint_disabled

Disabled task

3.9.2.2 bool srt_task_attr::constraint_forced

Forced run task

3.9.2.3 bool srt_task_attr::constraint_hard

Hard real time task

The documentation for this struct was generated from the following file:

- sart.h

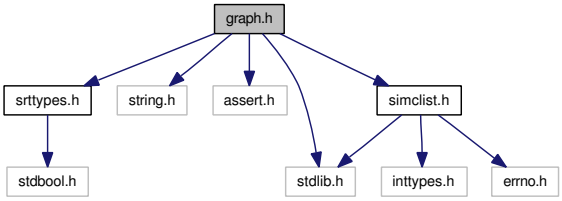
Chapter 4

File Documentation

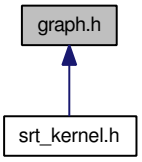
4.1 graph.h File Reference

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "srttypes.h"
#include "simclist.h"
```

Include dependency graph for graph.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **vertex**
- struct **edge**
- struct **graph**

Functions

- int graph_init (graph *g)
  initialize a graph object for use.

- void graph_destroy (graph *g)
- int graph_add_vertex (graph *g, void *data, SRT_CONCEPT type, const char *label)
- vertex * graph_vertex_at (graph *g, size_t index)
- vertex * graph_vertex_by_label (graph *g, const char *label)
- int graph_remove_vertex_at (graph *g, size_t index)
- int graph_remove_vertex (graph *g, vertex *v)
- int graph_add_edge (graph *g, vertex *v_source, vertex *v_dest, float w)
- int graph_remove_edge (graph *g, vertex *v_source, vertex *v_dest)
- int graph_remove_edge_at (graph *g, vertex *v, size_t edge_index)
- int graph_remove_all_edges (graph *g, vertex *v)
- edge * graph_edge_at (graph *g, vertex *v, size_t index)

4.1.1 Detailed Description

4.1.2 Function Documentation

4.1.2.1 int graph_add_edge (graph * *g*, vertex * *v_source*, vertex * *v_dest*, float *w*)

adding an edge to a vertex of graph

Parameters:

g a pointer to an initialized graph object
v_source adding an edge to this vertex
v_dest adding this edge to the vertex
w weight of the edge

Returns:

0 for success. other values for failure

adding an edge to an existed vertex

4.1.2.2 int graph_add_vertex (graph * *g*, void * *data*, SRT_CONCEPT *type*, const char * *label*)

adding a vertex to the graph

Parameters:

g a pointer to an initialized graph object
data a pointer to the vertex's relevant data
type type of the vertex
label label of the vertex

Returns:

0 for success. other values for failure

checking existence of the node

4.1.2.3 void graph_destroy (graph * *g*)

completely remove the graph from memory.

Parameters:

g a pointer to an initialized graph object

Returns:

nothing!

4.1.2.4 edge* graph_edge_at (graph * *g*, vertex * *v*, size_t *index*)

retrieving an edge of a vertex noted by its index

Parameters:

g a pointer to an initialized graph object
v the edge of the relevant vertex
index index of an edge in the vertex

Returns:

a pointer to the vertex for success. NULL for failure.

retrieving an edge of a vertex noted by its index

4.1.2.5 int graph_init (graph * *g*)

initialize a graph object for use.

Parameters:

g a pointer to a graph object

Returns:

0 for success. other values for failure

graph initializing

4.1.2.6 int graph_remove_all_edges (graph * *g*, vertex * *v*)

removing all edges from a vertex of graph denoted by v

Parameters:

g a pointer to an initialized graph object
v removing the edges from this vertex

Returns:

0 for success. other values for failure

removing all edges from a vertex of graph denoted by v

4.1.2.7 int graph_remove_edge (graph * *g*, vertex * *v_source*, vertex * *v_dest*)

removing an edge from a vertex of graph

Parameters:

g a pointer to an initialized graph object

v_source removing the edge from this vertex

v_dest corresponded edge to be removed

Returns:

0 for success. other values for failure

removing an edge from a vertex of graph

4.1.2.8 int graph_remove_edge_at (graph * *g*, vertex * *v*, size_t *edge_index*)

removing an edge from a vertex of graph by its index

Parameters:

g a pointer to an initialized graph object

v removing the edge from this vertex

edge_index corresponded edge by index

Returns:

0 for success. other values for failure

removing an edge from a vertex of graph by the index

4.1.2.9 int graph_remove_vertex (graph * *g*, vertex * *v*)

removing a vertex from the graph

Parameters:

g a pointer to an initialized graph object

v the vertex to be removed

Returns:

0 for success. other values for failure

removing a vertex from the graph

removing all the edges of g connected to vertex v

4.1.2.10 int graph_remove_vertex_at (graph * *g*, size_t *index*)

removing a vertex from the graph by index

Parameters:

g a pointer to an initialized graph object

index index of vertex in the graph

Returns:

0 for success. other values for failure

removing a vertex from the graph by index

removing all the edges of g connected to vertex v

4.1.2.11 vertex* graph_vertex_at (graph * *g*, size_t *index*)

retrieving a vertex of the graph by index

Parameters:

g a pointer to an initialized graph object

index index of vertex in the graph

Returns:

a pointer to the vertex for success. NULL for failure.

retrieving a vertex of the graph by index

4.1.2.12 vertex* graph_vertex_by_label (graph * *g*, const char * *label*)

retrieving a vertex of the graph by its label

Parameters:

g a pointer to an initialized graph object

label label of vertex in the graph

Returns:

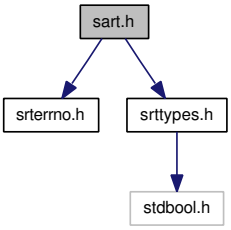
a pointer to the vertex for success. NULL for failure.

retrieving a vertex of the graph by its label

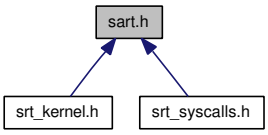
4.2 sart.h File Reference

```
#include "srterrno.h"
#include "srttypes.h"
```

Include dependency graph for sart.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct srt_task_attr
- struct srt_job_attr

Typedefs

- typedef bool(∗ schd_checker)(const float u, const u_int n)

Functions

- SRT_DATA ∗ srt_data_create (const char ∗label, const char ∗kind, u_int quality)
- srt_status srt_data_set_attr (SRT_DATA ∗data, u_int quality)
- SRT_RESOURCE ∗ srt_res_create (const char ∗label, bool available)
- srt_status srt_res_add_provision (SRT_RESOURCE ∗res, SRT_DATA ∗data)
- srt_status srt_res_set_attr (SRT_RESOURCE ∗res, bool available)
- SRT_JOB ∗ srt_job_create (const char ∗label, u_int prominence, SRT_POLICY policy)
- srt_status srt_job_add_data (SRT_JOB ∗job, SRT_DATA ∗data)
- srt_status srt_job_add_task (SRT_JOB ∗job, SRT_TASK ∗task)
- srt_status srt_job_set_attr (SRT_JOB ∗job, srt_job_attr attr)
- srt_status srt_job_get_attr (SRT_JOB ∗job, srt_job_attr ∗pattr)
- SRT_DISJUNCTION ∗ srt_disj_create (const char ∗label)
- srt_status srt_disj_add_data (SRT_DISJUNCTION ∗disj, SRT_DATA ∗data)
- srt_status srt_disj_add_disj (SRT_DISJUNCTION ∗disj, SRT_DISJUNCTION ∗disj2)
- srt_status srt_disj_add_conj (SRT_DISJUNCTION ∗disj, SRT_CONJUNCTION ∗conj)

- SRT_CONJUNCTION * srt_conj_create (const char *label)
- srt_status srt_conj_add_data (SRT_CONJUNCTION *conj, SRT_DATA *data)
- srt_status srt_conj_add_disj (SRT_CONJUNCTION *conj, SRT_DISJUNCTION *disj)
- srt_status srt_conj_add_conj (SRT_CONJUNCTION *conj, SRT_CONJUNCTION *conj2)
- SRT_TASK * srt_task_create (const char *label, u_int period, u_int consumption)
- srt_status srt_task_set_attr (SRT_TASK *task, srt_task_attr attr)
- srt_status srt_task_get_attr (SRT_TASK *task, srt_task_attr *pattr)
- srt_status srt_task_add_disj_input (SRT_TASK *task, SRT_DISJUNCTION *disj)
- srt_status srt_task_add_conj_input (SRT_TASK *task, SRT_CONJUNCTION *conj)
- srt_status srt_task_add_data_input (SRT_TASK *task, SRT_DATA *data)
- srt_status srt_task_add_output (SRT_TASK *task, SRT_DATA *data)
- srt_status srt_task_set_descriptor (SRT_TASK *task, void *descriptor)
- void * srt_get_entity_by_name (const char *name)
- srt_status srt_init (void)
- srt_status srt_finish (void)
- srt_status srt_chek_kb (void)
- srt_status srt_test_solution (void)
- srt_status srt_switch_context (void *os_data)
- srt_status srt_set_sched_checker (schd_checker check_func)
- srt_status srt_export_kb2dot (const char *filename)

4.2.1 Detailed Description

SART APIs for initialization, Knowledge base description and manipulation, context switching and scheduling.

4.2.2 Typedef Documentation

4.2.2.1 typedef bool(* schd_checker)(const float u, const u_int n)

User defined routine for schedulability test .

Parameters:

u utilization factor

n number of tasks

Returns:

true if it's feasible otherwise false.

4.2.3 Function Documentation

4.2.3.1 srt_status srt_chek_kb (void)

Checking for the consistency of the SART knowledge-base;

Returns:

ERR_OK if success

4.2.3.2 srt_status srt_conj_add_conj (SRT_CONJUNCTION * *conj*, SRT_CONJUNCTION * *conj2*)

Add CONJUNCTION entity to a DISJUNCTION

Parameters:

conj a pointer to SRT_CONJUNCTION structure

conj2 a pointer to SRT_CONJUNCTION structure

Returns:

ERR_OK if success

4.2.3.3 srt_status srt_conj_add_data (SRT_CONJUNCTION * *conj*, SRT_DATA * *data*)

Add DATA entity to a CONJUNCTION

Parameters:

conj a pointer to SRT_CONJUNCTION structure

data a pointer to SRT_DATA structure

Returns:

ERR_OK if success

4.2.3.4 srt_status srt_conj_add_disj (SRT_CONJUNCTION * *conj*, SRT_DISJUNCTION * *disj*)

Add DISJUNCTION entity to a CONJUNCTION

Parameters:

conj a pointer to SRT_CONJUNCTION structure

disj a pointer to SRT_DISJUNCTION structure

Returns:

ERR_OK if success

4.2.3.5 SRT_CONJUNCTION* srt_conj_create (const char * *label*)

Create a CONJUNCTION concept in SART Knowledge Base

Parameters:

label label

Returns:

a pointer to the SRT_CONJUNCTION structure if success; otherwise NULL.

4.2.3.6 SRT_DATA* srt_data_create (const char * *label*, const char * *kind*, u_int *quality*)

Create a DATA concept in SART Knowledge Base

Parameters:

label label
kind type of data
quality quality of data

Returns:

a pointer to the SRT_DATA structure if success; otherwise NULL.

4.2.3.7 srt_status srt_data_set_attr (SRT_DATA * *data*, u_int *quality*)

Set the attribute of a DATA

Parameters:

data a pointer to SRT_DATA structure
quality quality of data

Returns:

ERR_OK if success

4.2.3.8 srt_status srt_disj_add_conj (SRT_DISJUNCTION * *disj*, SRT_CONJUNCTION * *conj*)

Add CONJUNCTION entity to a DISJUNCTION

Parameters:

disj a pointer to SRT_DISJUNCTION structure
conj a pointer to SRT_CONJUNCTION structure

Returns:

ERR_OK if success

4.2.3.9 srt_status srt_disj_add_data (SRT_DISJUNCTION * *disj*, SRT_DATA * *data*)

Add DATA entity to a DISJUNCTION

Parameters:

disj a pointer to SRT_DISJUNCTION structure
data a pointer to SRT_DATA structure

Returns:

ERR_OK if success

4.2.3.10 srt_status srt_disj_add_disj (SRT_DISJUNCTION * *disj*, SRT_DISJUNCTION * *disj2*)

Add DISJUNCTION entity to a DISJUNCTION

Parameters:

disj a pointer to SRT_DISJUNCTION structure

disj2 a pointer to SRT_DISJUNCTION structure

Returns:

ERR_OK if success

4.2.3.11 SRT_DISJUNCTION* srt_disj_create (const char * *label*)

Create a DISJUNCTION concept in SART Knowledge Base

Parameters:

label label

Returns:

a pointer to the SRT_DISJUNCTION structure if success; otherwise NULL.

4.2.3.12 srt_status srt_export_kb2dot (const char * *filename*)

Export SART knowledge-base to a DOT graph file.

Parameters:

filename name of the output DOT file.

Returns:

ERR_OK if success.

4.2.3.13 srt_status srt_finish (void)

Ending SART;

Returns:

ERR_OK if success

NOTE: srt_finish() must be called at the end of you program.

4.2.3.14 void* srt_get_entity_by_name (const char * *name*)

Retrieve a SART entity by its name;

Parameters:

name label of the entity

Returns:

a pointer to a SART entity if success otherwise NULL;

NOTE: User must be aware of the type of the entity requested by its name!

4.2.3.15 srt_status srt_init (void)

Initializing SART;

Returns:

ERR_OK if success

NOTE: srt_init() must be called before any other SART APIs.

4.2.3.16 srt_status srt_job_add_data (SRT_JOB * *job*, SRT_DATA * *data*)

Add DATA necessity to a JOB

Parameters:

job a pointer to SRT_JOB structure

data a pointer to SRT_DATA structure

Returns:

ERR_OK if success

4.2.3.17 srt_status srt_job_add_task (SRT_JOB * *job*, SRT_TASK * *task*)

Add TASK necessity to a JOB

Parameters:

job a pointer to SRT_JOB structure

task a pointer to SRT_TASK structure

Returns:

ERR_OK if success

4.2.3.18 SRT_JOB* srt_job_create (const char * *label*, u_int *prominence*, SRT_POLICY *policy*)

Create a JOB concept in SART Knowledge Base

Parameters:

label label

prominence prominance or priority of JOB

policy ploicy concerning reasoning of job

Returns:

a pointer to the SRT_JOB structure if success; otherwise NULL.

4.2.3.19 srt_status srt_job_get_attr (SRT_JOB * *job*, srt_job_attr * *pattr*)

Set the attribute of Job

Parameters:

job a pointer to SRT_JOB structure

pattr a pointer to srt_job_attr structure to save the attributes

Returns:

ERR_OK if success

4.2.3.20 srt_status srt_job_set_attr (SRT_JOB * *job*, srt_job_attr *attr*)

Set the attribute of Job

Parameters:

job a pointer to SRT_JOB structure

attr srt_job_attr structure

Returns:

ERR_OK if success

4.2.3.21 srt_status srt_res_add_provision (SRT_RESOURCE * *res*, SRT_DATA * *data*)

Add provision to a RESOURCE

Parameters:

res a pointer to SRT_RESOURCE structure

data a pointer to SRT_DATA structure

Returns:

ERR_OK if success

4.2.3.22 SRT_RESOURCE* srt_res_create (const char * *label*, bool *available*)

Create a RESOURCE concept in SART Knowledge Base

Parameters:

label label

available availability of resource

Returns:

a pointer to the SRT_RESOURCE structure if success; otherwise NULL.

4.2.3.23 srt_status srt_res_set_attr (SRT_RESOURCE * *res*, bool *available*)

Set availability attribute of a resource

Parameters:

res a pointer to SRT_RESOURCE structure

available attribute of resource

Returns:

ERR_OK if success

4.2.3.24 srt_status srt_set_sched_checker (schd_checker *check_func*)

Set a user defined routine for schedulability test.

Parameters:

check_func a pointer to schd_checker function

Returns:

ERR_OK if success.

4.2.3.25 srt_status srt_switch_context (void * *os_data*)

Reason the current SART knowledge-base and switch to new context if any solution exists.

Parameters:

os_data OS dependent required data for low-level task manipulating. please refer to the document for the specific OS in the /arch folder.

Returns:

ERR_OK if success

4.2.3.26 srt_status srt_task_add_conj_input (SRT_TASK * *task*, SRT_CONJUNCTION * *conj*)

Add CONJUNCTION entity to the input of the task

Parameters:

task a pointer to SRT_TASK structure

conj a pointer to SRT_CONJUNCTION structure

Returns:

ERR_OK if success

4.2.3.27 srt_status srt_task_add_data_input (SRT_TASK * *task*, SRT_DATA * *data*)

Add DATA entity to the input of the task

Parameters:

task a pointer to SRT_TASK structure

data a pointer to SRT_DATA structure

Returns:

ERR_OK if success

4.2.3.28 srt_status srt_task_add_disj_input (SRT_TASK * *task*, SRT_DISJUNCTION * *disj*)

Add DISJUNCTION entity to the input of the task

Parameters:

task a pointer to SRT_TASK structure

disj a pointer to SRT_DISJUNCTION structure

Returns:

ERR_OK if success

4.2.3.29 srt_status srt_task_add_output (SRT_TASK * *task*, SRT_DATA * *data*)

Add Data entity to the output of a task

Parameters:

task a pointer to SRT_TASK structure

data a pointer to SRT_DATA structure

Returns:

ERR_OK if success

4.2.3.30 SRT_TASK* srt_task_create (const char * *label*, u_int *period*, u_int *consumption*)

Create a TASK concept in SART Knowledge Base

Parameters:

label label

period period of the periodic task

consumption task worst case execution time

Returns:

a pointer to the SRT_TASK structure if success; otherwise NULL.

4.2.3.31 srt_status srt_task_get_attr (SRT_TASK * *task*, srt_task_attr * *pattr*)

Retrieve the attribute of a TASK

Parameters:

task a pointer to SRT_TASK structure

pattr a pointer to srt_task_attr structure to save the attributes

Returns:

ERR_OK if success

4.2.3.32 srt_status srt_task_set_attr (SRT_TASK * *task*, srt_task_attr *attr*)

Set the attribute of TASK

Parameters:

task a pointer to SRT_TASK structure

attr srt_task_attr structure

Returns:

ERR_OK if success

4.2.3.33 srt_status srt_task_set_descriptor (SRT_TASK * *task*, void * *descriptor*)

Set the OS dependent task descriptor. This descriptor will be used by SART core to make relation between TASK concepts defined in the SART knowledge-base and real operating system tasks.

Parameters:

task a pointer to SRT_TASK structure

descriptor a pointer to a OS dependent task descriptor. for detailed information about the type and structure of this descriptor, please refer to the document for the specific OS in the /arch folder.

Returns:

ERR_OK if success

4.2.3.34 srt_status srt_test_solution (void)

Checks whether any solution exist in SART knowledge-base by the reasoner or not.

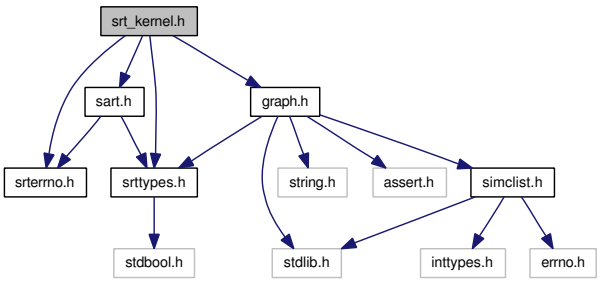
Returns:

ERR_OK if success

4.3 srt_kernel.h File Reference

```
#include "srttypes.h"
#include "srterrno.h"
#include "graph.h"
#include "sart.h"
```

Include dependency graph for srt_kernel.h:



Data Structures

- struct **srt_data_t**
- struct **srt_conjunction_t**
- struct **srt_disjunction_t**
- struct **srt_task_t**
- struct **srt_resource_t**
- struct **srt_job_t**
- struct **sart_kb**

Functions

- srt_status update_kb_graph (void)
- bool rm_sched_checker (const float u, const u_int n)
- srt_status select_task_set (void)
- float min_dependency_path (graph ∗g, vertex ∗initial, list_t ∗path)
- u_int count_common_task (list_t ∗l1, list_t ∗l2, SRT_CONCEPT rt)

Variables

- sart_kb **srt_kb**

4.3.1 Detailed Description

4.3.2 Function Documentation

4.3.2.1 u_int count_common_task (list_t ∗ *l1*, list_t ∗ *l2*, SRT_CONCEPT *rt*)

count the number of common tasks in two lists

4.3.2.2 float min_dependency_path (graph * *g*, vertex * *initial*, list_t * *path*)

TODO: 1) Needs to find the second and other feasibly schedulable task sets. 2) For optimization purpose, change all the list exploration by iteration method using list_iterator_start/next()

If this is an unavailable resource or a disabled task return with cost = -1 which says to not explore into this branch any more.

We are visiting this node. "visited" flag MUST always be cleared before returning from the routine

if this is a leaf, return with the cost = zero

Preventing getting stuck in a loop Do not meet a node which has already been visited.

A Soft-constrained task has no cost!!!

Explore deeper into graph by going toward one branch

The return cost of bottom leaves should be positive. A negative value means a task constraint or resource availability are not met.

sum up costs of the branches of a conjunction node.

chose the minimum branch.

In case of having two different path with the same costs, we should chose the path which most of its tasks has already been chosen for the other job.

TODO: Needs to be changed to support soft real-time tasks too.

An empty result occurs only and only if the constraint for the bottom leaves are not met. Therefore the current leaf should also be ignored.

adding initial job to the path

Adding tasks to the main path

4.3.2.3 bool rm_sched_checker (const float *u*, const u_int *n*)

Default schedubility checker routine (Rate Monotonic).

4.3.2.4 srt_status select_task_set (void)

Selecting the most suitable task set to perform user jobs. TODO: 1) MAX_QUALITY policy for the job has not been implemented. select_task_set() and min_dependency_path() need to be modified to support different job policies. 2) In order to have more efficient reasoner which allows to check all the combination of the tasks set and aims to schedule most possible hard real-time jobs, first it's needed to find all the possible dependency paths (task set) which are independently schedulable. Then try to find the combination of tasks which allow to schedule more jobs within hard real-time constraint.

checking for the consistency of the KB

defining a list to keep all the nodes in the minimum dependency path including data, con/disjunction and etc.

Adding all the Forced task to the list

If the task set is not feasibly schedulable return with failure!

finding minimum dependency tasks set for each job

saving the beginning position of next task set

negative cost means no feasible path is found! max_cost = cost + forced_cost.

if there is no feasible path for this job, remove it from the task set.

if no feasible path is found, return with failure

checking if its needed to update the task set.

need to update the task_set list

select tasks which are needed to be decativated

4.3.2.5 srt_status update_kb_graph (void)

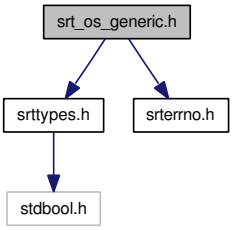
sart core internal routines

TODO: 1) two data input from different task should not be connected to each other!

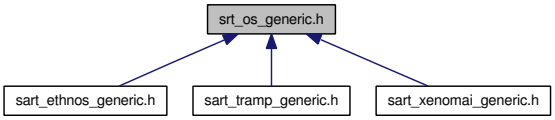
4.4 srt_os_generic.h File Reference

```
#include "srttypes.h"
#include "srterrno.h"
```

Include dependency graph for srt_os_generic.h:



This graph shows which files directly or indirectly include this file:



Functions

- void __pre_context_switch (void *os_data)
- void __post_context_switch (void *os_data)
- int __is_task_active (void *task_descriptor, void *os_data)
- int __activate_task (void *task_descriptor, void *os_data)
- int __deactivate_task (void *task_descriptor, void *os_data)

4.4.1 Detailed Description

4.4.2 Function Documentation

4.4.2.1 int __activate_task (void * *task_descriptor*, void * *os_data*)

This function will be called by sart to activate a task.

Parameters:

task_descriptor pointer to an os task descriptor.

os_data OS dependent data

Returns:

0 for success; a non-zero value for failure.

4.4.2.2 int __deactivate_task (void * *task_descriptor*, void * *os_data*)

This function will be called by sart to deactivate a task.

Parameters:

task_descriptor pointer to an os task descriptor.

os_data OS dependent data

Returns:

0 for success; a non-zero value for failure.

4.4.2.3 int __is_task_active (void * *task_descriptor*, void * *os_data*)

This function will be called by sart to indicate whether a task is activated or not.

Parameters:

task_descriptor pointer to an os task descriptor

os_data OS dependent data

Returns:

0 if the task is not activated; otherwise a non-zero value should be returned.

4.4.2.4 void __post_context_switch (void * *os_data*)

Finalizing context switch. this function will be called by sart context switch routine after activating/deactivating task sets. You might put any post processing and configuring OS.

Parameters:

os_data OS dependent data

4.4.2.5 void __pre_context_switch (void * *os_data*)

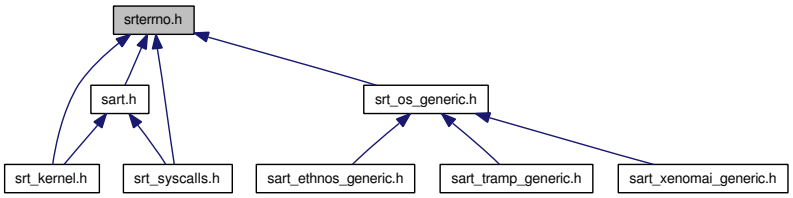
This header file includes the os (architecture) dependent routines which are needed to implemented as wrappers to SART in ./arch folder. Preparing for switching context. this function will be called by sart context switch routine before activating/deactivating task sets. You might put any initializing or configuring OS before actually switching to new context.

Parameters:

os_data OS dependent data

4.5 srterrno.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define __ASSERT(_cond)
- #define **__CHECK_NULLPTR**(_ptr) if(!_ptr) return ERR_NULL;
- #define **__DEBUG_MSG**(msg)
- #define **ERR_OK** 0
- #define **ERR_NULL** 1
- #define **ERR_FAILED** 2
- #define **ERR_NEED_UPDATE** 3
- #define ERR_NO_DATAKIND 100
- #define **ERR_NO_CONCITENCY** 101
- #define **ERR_NO_TASK_DESCRIPTOR** 102

4.5.1 Detailed Description

4.5.2 Define Documentation

4.5.2.1 #define __ASSERT(_cond)

Enable debug mode from here

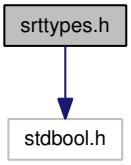
4.5.2.2 #define ERR_NO_DATAKIND 100

error codes representing knowledge base constraints

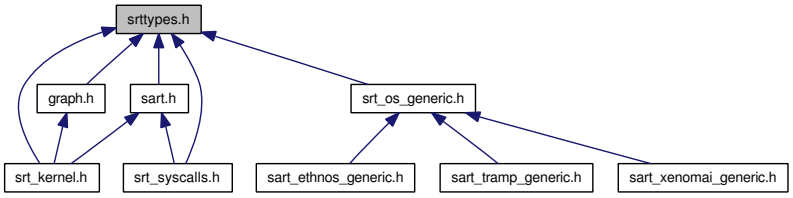
4.6 srttypes.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for srttypes.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct SRT_TASK
- struct SRT_DATA
- struct SRT_RESOURCE
- struct SRT_JOB
- struct SRT_CONJUNCTION
- struct SRT_DISJUNCTION

Defines

- #define **__sart_name__** "SART"
- #define **true** (int)1
- #define **false** (int)0
- #define **NULL** ((void∗)0)
- #define **DATA_KIND_SIZE** 128
- #define **LABLE_SIZE** 128
- #define SYSCALL(rettype) rettype
- #define OSROUTINE(rettype) rettype

Typedefs

- typedef unsigned int **bool**
- typedef unsigned char **byte**
- typedef unsigned char **u_char**
- typedef unsigned int **u_int**

- typedef unsigned long **u_long**
- typedef unsigned long long **u_llong**
- typedef u_int **srt_id**
- typedef u_int **srt_status**

Enumerations

- enum SRT_CONCEPT {
  CONSTRAINT, **DATA**, **TASK**, **RESOURCE**,
  JOB, **HARD**, **SOFT**, **FORCED**,
  DISABLED, **CONJUNCTION**, **DISJUNCTION** }
- enum SRT_POLICY { **MIN_COST**, **MAX_QUALITY** }

4.6.1 Detailed Description

4.6.2 Define Documentation

4.6.2.1 #define OSROUTINE(rettype) rettype

OS dependent routine

4.6.2.2 #define SYSCALL(rettype) rettype

SART system call routine

4.6.3 Enumeration Type Documentation

4.6.3.1 enum SRT_CONCEPT

SART KB related concepts

4.6.3.2 enum SRT_POLICY

SART reasoning policy

Index