

## NAME

smacq-embed — Embedding libsmacq in applications

## SYNOPSIS

```
#include <smacq.h>
```

## DESCRIPTION

SMACQ (the System for Modular Analysis and Continuous Queries) is an extensible component system for analyzing streams of structured data. This manpage describes the API used by applications that wish to use this system.

```
smacq_graph * smacq_build_pipeline(  
    int argc,  
    char ** argv);
```

Given an argument vector in the typical format, parse those arguments and create and return a data-flow graph. An argument of "|" is used to delimit modules in the pipeline.

```
smacq_graph * smacq_build_query(  
    dts_environment *,  
    int argc,  
    char ** argv);
```

Given an argument vector in the typical format, parse those arguments according to the SMACQ query language described in **smacq(1)** as a single string and create and return a data-flow graph.

```
smacq_graph * smacq_start(  
    smacq_graph * graph,  
    enum smacq_scheduler,  
    dts_environment * tenv);
```

Given the specified data-flow graph, instantiate the modules and begin the scheduler. Possible scheduler values are ITERATIVE, RECURSIVE, and THREADED. In the case of RECURSIVE or THREADED, smacq\_start() will never return. In the case of ITERATIVE, the scheduler must be repeatedly called using the following smacq\_sched\_iterative() function.

If tenv is NULL, a new type environment will automatically be created. If dts\_objects are to be passed between data-flow graphs, the same type environment must be used by those graphs.

```
int smacq_sched_iterative(  
    smacq_graph * graph,  
    const dts_object * data_in,  
    dts_object ** data_out,  
    struct runq ** runq,  
    int produce_first);
```

Execute the iterative scheduler until one data object is produced, or the system has terminated itself.

If data is produced, a pointer to it will be stored in data\_out. If the return value is 0, then smacq\_sched\_iterative() should not be called again.

If the caller wishes to inject data into the graph, a non-null `data_in` argument should be used.

The Boolean `produce_first` argument specifies that the head node of the graph can produce data when more is needed.

On the first call, the state argument should be a pointer to a void \* initialized to NULL.

```
int smacq_sched_iterative_shutdown(  
    smacq_graph * graph,  
    struct runq ** runq);
```

## **SEE ALSO**

`smacqp(1)`, `smacq-modules(3)`