

NAME

dts – Dynamic Type System API

SYNOPSIS

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

SMACQ(1) is built around a Dynamic Type System (DTS) runtime. The library routines used to interface with the DTS are described herein.

DESCRIPTION

SMACQ(1) is an extensible component system for analyzing streams of structured data. This manpage describes the programming API for creating pipeline modules. Type modules are documented separately in dts-modules(3).

The following routines provide access to data objects:

OBJECT CONSTRUCTION AND MODIFICATION

const dts_object * smacq_dts_construct(smacq_environment * env, int type, void * data)

Return a new data object with contents copied from data. This call only works for fixed-size objects.

const dts_object * smacq_alloc(smacq_environment * env, int datasize, int type)

Return a new, empty data object large enough to hold *datasize* bytes. Use *smacq_requiretype()* to get a type number.

int smacq_fromstring(smacq_environment * env, int type, char * value, dts_object * data)

Convert the string-form value into a typed data object of the specified type.

const dts_object * dts_construct_fromstring(dts_environment * env, int type, char * value)

Convert the string-form value into a new, typed data object of the specified type.

void dts_attach_field(const dts_object * current_object, dts_field field, const dts_object * field_data)

Attach the field_data object as a field of the current object.

void smacq_msg_send(smacq_environment * env, dts_field field, dts_object * data, dts_comparison * comparisonList)

Send a specified field with value specified by data to any data records described by the criteria in the comparison list.

const dts_object * dts_writable(smacq_environment * env, dts_object * obj)

Return a pointer to a writable version of the specified object. This operation invalidates the original object so it should not be used any further and should not be passed on to other modules.

OBJECT USE

int smacq_datum_size(const dts_object * datum)

int dts_getsize(const dts_object * d)

Return the size of the data

int dts_type_size(dts_environment * tenv, int type)

Return the size of a data type. -1 indicates a variable-size type. -2 indicates an error.

int dts_setsize(const dts_object * cd, int size)

void * dts_getdata(const dts_object * datum)
Return a pointer to the data

TYPE dts_data_as(const dts_object * datum, TYPE)

This is a macro that casts the data portion of the datum to the specified type. The result can be used either as r-value or an l-value.

void dts_set(const dts_object *, TYPE, VAL)

This is a macro that ensures that the object is large enough for the size of the specified type and then sets the object contents equal to the specified value.

int dts_gettype(const dts_object * datum)

Return the type of the data

int smacq_datum_settype(const dts_object * d, int type)

Set the type of the data

int dts_incref(const dts_object * datum, int num)

Increment the reference counter of the object by *num*.

int dts_decref(const dts_object * datum)

Decrement the reference counter of the object and free it if there are no more references.

OBJECT FIELD ACCESS

Data records are composed of named fields that can be accessed with the following functions:

const dts_object * smacq_getfield(smacq_environment * env, const dts_object * datum, dts_field field, dts_object * scratch)

const dts_object * dts_getfield(dts_environment * env, const dts_object * datum, dts_field field, dts_object * scratch)

Return a data object for the specified field of the specified object. The field object is only valid for the lifetime of the original object. `dts_decref()` must be used to free the object. NULL is returned on failure. The scratch argument should always be NULL.

const dts_object * smacq_getfield_copy(smacq_environment * env, const dts_object * datum, dts_field * scratch)

Same as above, but return persistent data that will outlive the datum. `dts_decref()` must be used to free the object. NULL is returned on failure. The scratch argument should be NULL.

There are also functions that provide access to groups of fields:

void fields_init(smacq_environment * env, struct fieldset * fieldset, int argc, char ** argv)

Given a string vector containing 1 or more field names, initialize the given fieldset vector.

struct iovec * fields2vec(smacq_environment * env, const dts_object * datum, struct fieldset * fieldset)

Returns an iovec for the specified fieldset in the specified object. If the object does not have one or more fields, then the iovec entry for that field will have a zero length. The size of the returned iovec is the same as fieldset->num.

int iovec_has_undefined(struct iovec *, int)

Return 1 iff the iovec has a zero-length element. Otherwise return 0.

RUNTIME ACCESS

dts_field smacq_requirefield(smacq_environment * env, char * tname)

dts_field dts_requirefield(dts_environment * env, char * tname)

Return the dynamically assigned identifier for the given field name. This dts_field must be freed with dts_field_free().

void dts_field_free(dts_field field)

Free the given dts_field data structure.

char * dts_fieldname_append(const char * base, const char * addition)

Return a newly allocated string containing the addition string appended to the base string and separated by a period.

int dts_comparefields(dts_field, dts_field)

Compare the two field specifications and return 0 unless they are the same.

int smacq_requiretype(smacq_environment * env, char * type_name)

int dts_requiretype(dts_environment * env, char * type_name)

Types are dynamically loaded classes. Load the specified type module (if it is not already loaded) and return the dynamically assigned numeric identifier for that type. This number will be consistent for the duration of this instantiation. It is recommended for performance that modules convert type names to integers sparingly and cache results. All datum structures are typed with these values.

int smacq_tynum_byname(smacq_environment * env, char * name)

If the specified type module is already loaded, this result is the same as requiretype(). Unlike requiretype(), if the type is not loaded, -1 is returned.

char * dts_tynum_byname(smacq_environment * env, int num)

Returns the string name of the specified numeric type identifier.

OBJECT COMPARISON

The system knows how to compare typed objects using the following routines. The basic data structure for this is dts_comparison. This structure has an operation type that may be one of EQUALITY, INEQUALITY, LIKE, GT, LT, AND, and OR. In the case of AND and OR, the "group" element points to a dts_comparison list of subterms. Otherwise, the value string is in the "valstr" element.

int smacq_match(smacq_environment * env, const dts_object * datum, dts_comparison * comps)

Compares the specified datum with the specified list of comparisons. Returns non-zero iff all of the comparisons are true.

dts_comparison * dts_parse_tests(dts_environment * tenv, int argc, char ** argv)

Return the comparison(s) resulting from the given argument vector. Comparisons can include AND and OR statements, parentheses for grouping, and equality and inequality operators.

SEE ALSO

smacq(1), smacqq(1), smacqp(1), dts-modules(3) smacq-modules(3)