

NAME

dts-modules — type module programming guide

SYNOPSIS

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

DESCRIPTION

SMACQ(1) is an extensible component system for analyzing streams of structured data. This manpage describes how type modules are added to the system.

By convention, types are compiled to an ELF shared library with a name like **smacqtype_x.so**, where **x** is the type name. Types modules are not containers for data, but merely interfaces for accessing data of that type. Thus there is no private storage associated with a type.

Many SMACQ *compute modules* are polymorphic and have no knowledge of the internal representation of typed data. They access the data through field and transform functions described below. However, a type-specific compute module may have this knowledge. Therefore, a type module that is expected to be used by type-specific modules should be accompanied by an include file defining the structure of the data. Type-specific compute modules may then use this include file.

dts_type_typename_table

Each type module must declare a structure named **dts_type_typename_table** of type **struct dts_type_info**, which is defined as follows:

```
struct dts_type_info {
    int size;
    smacqtype_lt_fn * lt;
    smacqtype_fromstring_fn * fromstring;
};
```

The size is the constant size, in bytes, of the data type (e.g. 1 for a char), or -1 if the data has variable size. The less-than and fromstring functions are optional and have the following prototypes:

```
typedef int smacqtype_lt_fn(void *, int, void *, int);

typedef int smacqtype_fromstring_fn(char *, const dts_object *);
```

dts_type_typename_fields

Each type module may declare a list named **dts_type_typename_fields** of **struct dts_field_spec** elements, which are defined as follows:

```
struct dts_field_spec {
    char * type;
    char * name;
    field_getfunc_fn * getfunc;
};

typedef int field_getfunc_fn(const dts_object*, dts_object *);
```

Each entry in this list describes a named field in a list of elements of this type. For example, the type may be a structure. The **type** string is the type of the element named **name**.

If the **getfunc** pointer is NULL, then it is assumed that this field occurs in the data immediately following

the previous field. The fixed-size fields in a C structure can be listed this way (but if you're describing a C struct, be aware of padding that the compiler may add to properly align C struct).

If **getfunc** is non-NULL, then it represents a function to be called to get initialize an object for this field. Variable sized fields must provide a getfunc. For fixed-length types, the data object will already be the correct size. Otherwise, the length should be set appropriately and the data pointer set to a new buffer of the same size. If a new buffer is allocated, the "free_data" element should be set to indicate that the buffer should be freed along with the data object.

Field access functions

```
int field_getfunc_fn(  
    const dts_object * datum,  
    dts_object * field_datum);
```

Set ***data** to point to the field in **datum**. Set ***len** to the correct length. Return 1 on success or 0 if the field is not present in this datum.

dts_type_tynename_transforms

Each type module may declare a list named **dts_type_tynename_transforms** of **struct dts_transform_descriptor** elements, which are defined as follows:

```
struct dts_field_spec {  
    char * name;  
    transform_getfunc_fn * getfunc;  
};
```

```
typedef int transform_getfunc_fn(void * data, int len, void ** tdata, int * tlen);
```

Each entry in this list describes a named transformation that can be applied to data of this type. Each type should provide a transform to a "string" so that the type can be printed. The transform is performed by calling **getfunc**. Transforms allocate storage for the transformation and the caller must free this space when it is done with the transform.

Field transform functions

```
int transform_getfunc_fn(  
    void * data,  
    int len,  
    void ** tdata,  
    int *tlen);
```

Allocate new storage for a transformation of the data of length **len** pointed to by ***data**. Return a pointer in ***tdata** and the transformed size in ***tlen**. Return 1 on success or 0 if the field is not present in this datum.

SEE ALSO

smacqp(1), smacq-modules(3)