NAME

Cdt – container data types

SYNOPSIS

```
#include <cdt.h>
```

DICTIONARY TYPES

```
Dt_t;
Dtdisc_t;
Dtmethod_t;
Dtlink_t;
Dtstat_t;
```

DICTIONARY CONTROL

```
Dt_t*       dtopen(const Dtdisc_t* disc, const Dtmethod_t* meth);
int         dtclose(Dt_t* dt);
void        dtclear(dt);
Dtmethod_t* dtmethod(Dt_t* dt, const Dtmethod_t* meth);
Dtdisc_t*   dtdisc(Dt_t* dt, const Dtdisc_t* disc);
Dt_t*       dtview(Dt_t* dt, Dt_t* view);
```

STORAGE METHODS

```
Dtmethod_t* Dtset;
Dtmethod_t* Dtoset;
Dtmethod_t* Dtobag;
```

DISCIPLINE

```
#define DTDISC(disc,key,size,link,makef,freef,comparf)
typedef void*      (*Dtmake_f)(void*, Dtdisc_t*);
typedef void        (*Dtfree_f)(void*);
typedef int        (*Dtcompar_f)(void*, void*);
```

OBJECT OPERATIONS

```
void*   dtinsert(Dt_t* dt, void* obj);
void*   dtdelete(Dt_t* dt, void* obj);
void*   dtdetach(Dt_t* dt, void* obj);
void*   dtsearch(Dt_t* dt, void* obj);
void*   dtmatch(Dt_t* dt, void* key);
void*   dtfirst(Dt_t* dt);
void*   dtnext(Dt_t* dt, void* obj);
void*   dtlast(Dt_t* dt);
void*   dtprev(Dt_t* dt, void* obj);
void*   dtfinger(Dt_t* dt);
void*   dtrenew(Dt_t* dt, void* obj);
int     dtwalk(Dt_t* dt, int (*userf)(void*, void*), void*);
Dtlink_t* dtflatten(Dt_t* dt);
Dtlink_t* dtlink(Dt_t*, Dtlink_t* link);
void*   dtobj(Dt_t* dt, Dtlink_t* link);
Dtlink_t* dtextract(Dt_t* dt);
int     dtrestore(Dt_t* dt, Dtlink_t* link);
```

DICTIONARY STATUS

```
int     dtsize(Dt_t* dt);
int     dtstat(Dt_t* dt, Dtstat_t*, int all);
```

HASH FUNCTIONS

```
unsigned int dtstrhash(void *str, int n);
```

DESCRIPTION

Cdt manages run-time dictionaries using standard container data types: unordered set/multiset, ordered set/multiset, list, and stack.

DICTIONARY TYPES

Dt_t
This is the type of a dictionary handle.

Dtdisc_t
This defines the type of a discipline structure which describes object lay-out and manipulation functions.

Dtmethod_t
This defines the type of a container method.

Dtlink_t
This is the type of a dictionary object holder (see dtdisc().)

Dtstat_t
This is the type of a structure to return dictionary statistics (see dtstat().)

DICTIONARY CONTROL

Dt_t* dtopen(const Dtdisc_t* disc, const Dtmethod_t* meth)
This creates a new dictionary. disc is a discipline structure to describe object format. meth specifies a manipulation method. dtopen() returns the new dictionary or NULL on error.

int dtclose(Dt_t* dt)
This deletes dt and its objects. Note that dtclose() fails if dt is being viewed by some other dictionaries (see dtview()). dtclose() returns 0 on success and -1 on error.

void dtclear(Dt_t* dt)
This deletes all objects in dt without closing dt.

Dtmethod_t dtmethod(Dt_t* dt, const Dtmethod_t* meth)
If meth is NULL, dtmethod() returns the current method. Otherwise, it changes the storage method of dt to meth. Switching to and from Dtset and Dtoset/Dtobag may cause objects to be rehashed, reordered, or removed as the case requires. dtmethod() returns the previous method or NULL on error.

Dtdisc_t* dtdisc(Dt_t* dt, const Dtdisc_t* disc)
If disc is NULL, dtdisc() returns the current discipline. Otherwise, it changes the discipline of dt to disc. Objects may be rehashed, reordered, or removed as appropriate. dtdisc() returns the previous discipline on success and NULL on error.

Dt_t* dtview(Dt_t* dt, Dt_t* view)
A viewpath allows a search or walk starting from a dictionary to continue to another. dtview() first terminates any current view from dt to another dictionary. Then, if view is NULL, dtview returns the terminated view dictionary. If view is not NULL, a viewpath from dt to view is established. dtview() returns dt on success and NULL on error.

It is an error to have dictionaries on a viewpath with different storage methods. In addition, dictionaries on the same view path should treat objects in a consistent manner with respect to comparison or hashing. If not, undefined behaviors may result.

STORAGE METHODS

Storage methods are of type Dtmethod_t*. *Cdt* supports the following methods:

Dtoset
Dtobag
Objects are ordered by comparisons. Dtoset keeps unique objects. Dtobag allows repeatable objects.

Dtset
Objects are unordered. Dtset keeps unique objects. This method uses a hash table with chaining to manage the objects.

DISCIPLINE

Object format and associated management functions are defined in the type Dtdisc_t:

```
typedef struct
{ int       key, size;
  int       link;
  Dtmake_f  makef;
  Dtfree_f  freef;
  Dtcompar_f comparf;
} Dtdisc_t;
```

int key, size

Each object obj is identified by a key used for object comparison or hashing. key should be non-negative and defines an offset into obj. If size is negative, the key is a null-terminated string with starting address *(void**)((char*)obj+key). If size is zero, the key is a null-terminated string with starting address (void*)((char*)obj+key). Finally, if size is positive, the key is a byte array of length size starting at (void*)((char*)obj+key).

int link

Let obj be an object to be inserted into dt as discussed below. If link is negative, an internally allocated object holder is used to hold obj. Otherwise, obj should have a Dtlink_t structure embedded link bytes into it, i.e., at address (Dtlink_t*)((char*)obj+link).

void* (*makef)(void* obj, Dtdisc_t* disc)

If makef is not NULL, dtinsert(dt,obj) will call it to make a copy of obj suitable for insertion into dt. If makef is NULL, obj itself will be inserted into dt.

void (*freef)(void* obj)

If not NULL, freef is used to destroy data associated with obj.

int (*comparf)(void* key1, void* key2)

If not NULL, comparf is used to compare two keys. Its return value should be <0, =0, or >0 to indicate whether key1 is smaller, equal to, or larger than key2. All three values are significant for method Dtoset and Dtobag. For other methods, a zero value indicates equality and a non-zero value indicates inequality. If (*comparf)() is NULL, an internal function is used to compare the keys as defined by the Dtdisc_t.size field.

#define DTDISC(disc,key,size,link,makef,freef,comparf)

This macro function initializes the discipline pointed to by disc with the given values.

OBJECT OPERATIONS

void* dtinsert(Dt_t* dt, void* obj)

This function adds an object prototyped by obj into dt. dtinsert() performs the same function for all methods. If there is an existing object in dt matching obj and the storage method is Dtset or Dtoset, dtinsert() will simply return the matching object. Otherwise, a new object is inserted according to the method in use. See Dtdisc_t.makef for object construction. The new object or a matching object as noted will be returned on success while NULL is returned on error.

void* dtdelete(Dt_t* dt, void* obj)

If obj is not NULL, there are two cases. If the method in use is not Dtobag, the first object matching obj is deleted. On the other hand, if the method in use is or Dtobag, the library check to see if obj is in the dictionary and delete it. If obj is not in the dictionary, some object matching it will be deleted. See Dtdisc_t.freef for object destruction. dtdelete() returns the deleted object (even if it was deallocated) or NULL on error.

void* dtdetach(Dt_t* dt, void* obj)

This function is similar to dtdelete() but the object to be deleted from dt will not be freed (via the discipline freef function).

void* dtsearch(Dt_t* dt, void* obj)

void* dtmatch(Dt_t* dt, void* key)
These functions find an object matching obj or key either from dt or from some dictionary accessible from dt via a viewpath (see dtview().) dtsearch() and dtmatch() return the matching object or NULL on failure.

void* dtfirst(Dt_t* dt)
void* dtnext(Dt_t* dt, void* obj)
dtfirst() returns the first object in dt. dtnext() returns the object following obj. Objects are ordered based on the storage method in use. For Dtoset and Dtobag, objects are ordered by object comparisons. For Dtset, objects are ordered by some internal order (more below). Thus, objects in a dictionary or a viewpath can be walked using a for(;;) loop as below.

```
for(obj = dtfirst(dt); obj; obj = dtnext(dt,obj))
```

When a dictionary uses Dtset, the object order is determined upon a call to dtfirst()/dtlast(). This order is frozen until a call dtnext()/dtprev() returns NULL or when these same functions are called with a NULL object argument. It is important that a dtfirst()/dtlast() call be balanced by a dtnext()/dtprev() call as described. Nested loops will require multiple balancing, once per loop. If loop balancing is not done carefully, either performance is degraded or unexpected behaviors may result.

void* dtlast(Dt_t* dt)
void* dtprev(Dt_t* dt, void* obj)
dtlast() and dtprev() are like dtfirst() and dtnext() but work in reverse order. Note that dictionaries on a viewpath are still walked in order but objects in each dictionary are walked in reverse order.

void* dtfinger(Dt_t* dt)
This function returns the *current object* of dt, if any. The current object is defined after a successful call to one of dtsearch(), dtmatch(), dtinsert(), dtfirst(), dtnext(), dtlast(), or dtprev(). As a side effect of this implementation of *Cdt*, when a dictionary is based on Dtoset and Dtobag, the current object is always defined and is the root of the tree.

void* dtrenew(Dt_t* dt, void* obj)
This function repositions and perhaps rehashes an object obj after its key has been changed. dtrenew() only works if obj is the current object (see dtfinger()).

dtwalk(Dt_t* dt, int (*userf)(void*, void*), void* data)
This function calls (*userf)(obj,data) on each object in dt and other dictionaries viewable from it. If userf() returns a <0 value, dtwalk() terminates and returns the same value. dtwalk() returns 0 on completion.

Dtlink_t* dtflatten(Dt_t* dt)
Dtlink_t* dtlink(Dt_t* dt, Dtlink_t* link)
void* dtobj(Dt_t* dt, Dtlink_t* link)
Using dtfirst()/dtnext() or dtlast()/dtprev() to walk a single dictionary can incur significant cost due to function calls. For efficient walking of a single directory (i.e., no viewpathing), dtflatten() and dtlink() can be used. Objects in dt are made into a linked list and walked as follows:

```
for(link = dtflatten(dt); link; link = dtlink(dt,link) )
```

Note that dtflatten() returns a list of type Dtlink_t*, not void*. That is, it returns a dictionary holder pointer, not a user object pointer (although both are the same if the discipline field link is zero.) The macro function dtlink() returns the dictionary holder object following link. The macro function dtobj(dt,link) returns the user object associated with link, Beware that the flattened object list is unflattened on any dictionary operations other than dtlink().

Dtlink_t* dtextract(Dt_t* dt)
int dtrestore(Dt_t* dt, Dtlink_t* link)
dtextract() extracts all objects from dt and makes it appear empty. dtrestore() repopulates dt with objects previously obtained via dtextract(). dtrestore() will fail if dt is not empty. These functions can be used to share a same dt handle among many sets of objects. They are useful to reduce dictionary overhead in an application that creates many concurrent dictionaries. It is important that the same discipline and method are in use at both extraction and restoration. Otherwise, undefined behaviors may result.

DICTIONARY INFORMATION

int dtsize(Dt_t* dt)

This function returns the number of objects stored in dt.

int dtstat(Dt_t *dt, Dtstat_t* st, int all)

This function reports dictionary statistics. If all is non-zero, all fields of st are filled. Otherwise, only the dt_type and dt_size fields are filled. It returns 0 on success and -1 on error.

Dtstat_t contains the below fields:

int dt_type:
: This is one of DT_SET, DT_OSET, and DT_OBAG.

int dt_size:
: This contains the number of objects in the dictionary.

size_t dt_n:
: For Dtset, this is the number of non-empty chains in the hash table. For Dtoset and Dtobag, this is the deepest level in the tree (counting from zero.) Each level in the tree contains all nodes of equal distance from the root node. dt_n and the below two fields are undefined for other methods.

size_t dt_max:
: For Dtset, this is the size of a largest chain. For Dtoset and Dtobag, this is the size of a largest level.

size_t *dt_count:
: For Dtset, this is the list of counts for chains of particular sizes. For example, dt_count[1] is the number of chains of size 1. For Dtoset and Dtobag, this is the list of sizes of the levels. For example, dt_count[1] is the size of level 1.

HASH FUNCTIONS

unsigned int dtstrhash(void *str, int n)

This function computes hash values from bytes or strings. dtstrhash() computes a new hash value from string str. If n is positive, str is a byte array of length n; otherwise, str is a null-terminated string.

IMPLEMENTATION NOTES

Dtset are based on hash tables with move-to-front collision chains. Dtoset and Dtobag are based on top-down splay trees.

AUTHOR

Kiem-Phong Vo, kpv@research.att.com