OSF DCE SIG
Request For Comments: 47.2

R. Salz (OSF)
P. Wang (OSF)
January 1994

DCED: THE DCE HOST DAEMON — PROGRAMMER'S API

1. INTRODUCTION

The DCED API provides a convenience library built on top of the IDL interfaces exported by `dced` [RFC 47.0] (see also [RFC 47.1]). The usage model is the same as that used elsewhere in DCE:

(a) A binding is created that is presented to the programmer as an opaque handle. There may be more than one way to create the binding.

(b) The binding is used as the first parameter in a set of domain-specific routines. These routines are called to do the ''real work'' required by the application.

(c) A routine is called to release the binding and its associated resources.

All routines described here also follow the same three-part calling sequence: input parameters, output parameters, and on output status parameter.

We are modifying the `dced` specification to use the Extended Registry Attributes being developed as part of the DCE 1.1 security work [RFC 6]. The redesign is not complete, so the calling sequence of two routines will be changing; comments in the declarations indicate the area of change.

Wherever possible, the DCED API provides generic routines rather than service-specific routines. For example, a single set of three routines is used to iterate over any set of `dced` objects, whether they are key tables or running servers. We believe this makes things simpler, at the cost of some type-safety.

All constants, datatypes, and routines are declared in the following header file:

```
#include <dce/dced.h>
```

For clarity, we will use IDL syntax in the declarations below.

2. BINDING AND IDENTIFCATION

The following routines are available to create and release `dced` bindings:

```
/* [opaque] */
typedef struct dced_binding_handle_s_t *dced_binding_handle_t;

void
dced_binding_create(
        [in]    dced_string_t           name,
        [in]    unsigned32              flags,
        [out]   dced_binding_handle_t   *h,
        [out]   error_status_t          *st
);
```

```
void
dced_binding_from_rpc_binding(
        [in]    dced_string_t          name,
        [in]    handle_t               rpc_handle,
        [out]   dced_binding_handle_t  *h,
        [out]   error_status_t         *st
);

void
dced_binding_free(
        [in]    dced_binding_handle_t  h,
        [out]   error_status_t         *st
);
```

The `name` parameter specifies both the desired host, and the service to be used for the generic routines. It can take any of the following forms:

> /.../*cellname*/*host*/config/*service*
> /.:/*host*/config/*service*
> *service*@*host*
> *service*

The first two forms identify a host in the global namespace or the local cell, respectively. The third form is a shorthand version of the second form. The last form is used to refer to either the local cell (for `dced_binding_create`) or to identify the service if using an existing RPC binding handle (`dced_binding_from_rpc_binding`). The `service` names the `dced` service and should be one of those services — `hostdata`, `srvrexec`, and so on.

Every object maintained by `dced` is identified by a UUID and a name (the server enforces unique names). The following routines can be used to map from one to the other:

```
void
dced_inq_id(
        [in]    dced_binding_handle_t  h,
        [in]    dced_string_t          name,
        [out]   uuid_t                 *id,
        [out]   error_status_t         *st
);

void
dced_inq_name(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [out]   dced_string_t          *name,
        [out]   error_status_t         *st
);
```

3. OBJECT LISTS

Each `dced` service maintains a list of its objects, this list is called the `entry list`. Conceptually, the list is the ''names'' of the objects, not the objects themselves. For the `srvrconf` and `srvrexec` services, the objects are the in-core `server_t` data, while for the `hostdata` and `rkeytab` services the objects are the data stored in the location specified by the `storage_tag` fields.

There are two ways to examine an entry list: the entire list may be retrieved at once, or a cursor may be used to step through an entry at a time. The following routines provide these capabilities:

```
void
dced_list_get(
        [in]    dced_binding_handle_t  h,
        [out]   dced_entry_list_t      *list
        [out]   error_status_t         *st
);

void
dced_list_release(
        [in]    dced_binding_handle_t  h,
        [out]   dced_entry_list_t      *list
        [out]   error_status_t         *st
);

/* [opaque] */
typedef struct dced_cursor_s_t *dced_cursor_t;

void
dced_cursor_initialize(
        [in]    dced_binding_handle_t  h,
        [out]   dced_cursor_t          *cursor,
        [out]   error_status_t         *st
);

void
dced_entry_get_next(
        [in]    dced_cursor_t          cursor,
        [out]   dced_entry_t           **entry,
        [out]   error_status_t         *st
);

void
dced_cursor_release(
        [in]    dced_cursor_t          cursor,
        [out]   error_status_t         *st
);
```

The ‘‘cursor’’ routines are provided so that applications can be written that do not require the entire entry list to be transmitted in a single RPC.

For the `hostdata` and `rkeytab` services it is possible to directly manipulate the entry list:

```
void
dced_entry_add(
        [in]    dced_binding_handle_t  h,
        [in]    dced_entry_t           *entry,
        [out]   error_status_t         *st
);

void
dced_entry_remove(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [out]   error_status_t         *st
);
```

The entry lists provided by the `srvrconf` and `srvrexec` cannot be directly manipulated by these routines.

4. RETRIEVING DATA

The following two routines may be used to read the object data:

```
void
dced_object_read_all(
        [in]    dced_binding_handle_t  h,
        [out]   unsigned32             *count,
        [out]   void                   **list
        [out]   error_status_t         *st
);

void
dced_object_read(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [out]   void                   **data,
        [out]   error_status_t         *st
);
```

Like the entry list routines, the first returns all objects, while the second returns the objects one at a time.

The following single routine frees either an object a list of object data:

```
void
dced_objects_release(
        [in]    dced_binding_handle_t  h,
        [in]    unsigned32             *count,
        [in]    void                   *dataptr
        [out]   error_status_t         *st
);
```

The following examples show how the routine defined above could be used:

```
dced_binding_handle     bh;
error_status_t          st;
dced_cursor_t           curs;
dced_entry_t            *e;
dced_key_list_t         *ktab;
server_t                *serv_array;

dced_binding_create("rkeytab", &bh, &st);
dced_cursor_initialize(bh, &curs, &st);
for ( ; ; ) {
    dced_entry_get_next(curs, &e, &st);
    if (st == dced_st_no_more)
        break;
    dced_object_read(bh, &e->uuid, (void *)&ktab, &st);
    dced_objects_release(bh, 1, (void *)ktab, &st);
}
dced_cursor_release(curs, &st);
dced_binding_free(bh, &st);
```

```
dced_binding_create("/.:/hosts/sulphur/config/srvrexec",
    &bh, &st);
dced_object_read_all(bh, &count, (void **)&serv_array, &st);
for (i = 0; i < count; i++)
    display(&serv_array[i]);
dced_objects_release(bh, count, (void *)serv_array, &st);
dced_binding_free(bh, &st);
```

5. SERVER ROUTINES

The following two routines create and delete `dced srvrconf` objects:

```
void
dced_server_create(
        [in]    dced_binding_handle_t  h,
        [in]    server_t               *object,
        [out]   error_status_t         *st
);

void
dced_server_delete(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [out]   error_status_t         *st
);
```

The following two routines can be used to start a server that has been configured with `dced`:

```
void
dced_server_start(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        /* <<Attributes subject to change after ERA review>> */
        [in]    dced_attr_list_t       attributes,
        [out]   error_status_t         *st
);

void
dced_server_stop(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [in]    srvrexec_stop_method   method,
        [out]   error_status_t         *st
);
```

If the `id` parameter given to `dced_server_stop` is a NULL pointer, then all servers on that host (including `dced` itself) are shut down.

The following two routines may be used to control what services are active within a server:

```
void
dced_server_enable_if(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [in]    rpc_if_id_t            *interface,
        [out]   error_status_t         *st
);
```

```
void
dced_server_disable_if(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [in]    rpc_if_id_t            *interface,
        [out]   error_status_t         *st
);
```

The following routine modifies the data of a server, either configured or running:

```
void
dced_server_modify_attributes(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        /* <<Attributes subject to change after ERA review>> */
        [in]    dced_attr_list_t       attributes,
        [out]   error_status_t         *st
);
```

6. HOSTDATA

The following two routines create and delete host data configuration items. They also cause the `hostdata` entry list to be updated:

```
void
dced_hostdata_create(
        [in]    dced_binding_handle_t  h,
        [in]    dced_entry_t           *entry,
        [in]    dced_attr_list_t       *data,
        [out]   error_status_t         *st
);

void
dced_hostdata_delete(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [out]   error_status_t         *st
);
```

Each `hostdata` item has attributes that contain the ‘‘real’’ data of the item. The following two routines read and write attributes. They are subject to change following review of the ERA spec.

```
void
dced_hostdata_read(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [out]   dced_attr_list_t       *hd_data,
        [out]   error_status_t         *st
);

void
dced_hostdata_write(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [in]    dced_attr_list_t       *hd_data,
        [out]   error_status_t         *st
);
```

Writing data replaces any existing value. The most common and useful attribute is the `filecontents` attribute which represents the file contents as an array of strings. This will have a well-known UUID.

7. RKEYTAB

The following two routines create and delete key tables. They also cause the `keytable` entry list to be updated:

```
void
dced_keytab_create(
        [in]    dced_binding_handle_t  h,
        [in]    dced_entry_t           *entry,
        [in]    dced_key_list_t        *keys,
        [out]   error_status_t         *st
);

void
dced_keytab_delete(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [out]   error_status_t         *st
);
```

The ‘‘object read’’ routines described above will retrieve the contents of a key table. To retrieve the keys for just a single principal, the following routine can be used:

```
void
dced_keytab_read_keys(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [in]    dced_string_t          principal,
        [out]   dced_key_list_t        *keys,
        [out]   error_status_t         *st
);
```

We expect that key tables will be small enough that there is no need for ‘‘cursor’’ routines.

The following routines add and delete keys from a key table:

```
void
dced_keytab_add_key(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [in]    dced_key_t             *key,
        [out]   error_status_t         *st
);

void
dced_keytab_delete_key(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [in]    dced_key_t             *key,
        [out]   error_status_t         *st
);
```

When deleting a key, if the key ‘‘version’’ is zero, then all keys for the specified principal are deleted.

It is also possible to change a key, updating both the registry and the key table, by calling the following routine:

```
void
dced_keytab_change_key(
        [in]    dced_binding_handle_t  h,
        [in]    uuid_t                 *id,
        [in]    dced_key_t             *key,
        [out]   error_status_t         *st
);
```

8. SECVAL

The following routines can be used to start or stop the security validation service within `dced`:

```
void
dced_secval_start(
        [in]    dced_binding_handle_t  h,
        [out]   error_status_t         *st
);

void
dced_secval_stop(
        [in]    dced_binding_handle_t  h,
        [out]   error_status_t         *st
);
```

The following routine may be used to validate the credentials returned by a DCE security server. It is most useful when `h` is bound to the local host and the program needs to complete the trust chain:

```
void
dced_secval_validate(
        [in]    dced_binding_handle_t  h,
        [out]   error_status_t         *st
);
```

9. ACKNOWLEDGEMENTS

John Shirley commented on early drafts and made many helpful suggestions on routine names and organization.

REFERENCES

[RFC 6] J. Pato, DCE-RFC 6.0, ‘‘A Generic Interface for Extended Registry Attributes’’, June 1992.

[RFC 47.0] D. Mackey, R. Salz, DCE-RFC 47.0, ‘‘DCED: The DCE Host Daemon — Functional Specification’’, December 1993.

[RFC 47.1] R. Salz, P. Wang, DCE-RFC 47.1, ‘‘DCED: The DCE Host Daemon — Command Set’’, January 1994.

AUTHORS’ ADDRESSES

Rich Salz
Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142
USA

Internet email: rsalz@osf.org
Telephone: +1-617-621-7253

Ping Wang
Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142
USA

Internet email: pwang@osf.org
Telephone: +1-617-621-8988