bound association. This is done using the appropriate routine for the required operation.

If operations are invoked asynchronously, then a call to the procedure `DapInitWaitRequest` will need to check for responses to outstanding operations and to return such a response as an indication if there is one available. The DUA can then process responses to the operations it invoked.

When the DUA no longer needs to be bound to a DSA, the procedure `DapUnBindRequest` can be used to initiate the termination of a bound association, and if necessary, the `DapUnBindRetry` routine used to complete the binding.

Chapter 19

Caching in a DUA

This part of the manual is written for implementors who wish to perform caching in their DUA. This used to require the use of the the *libquipu*(3n) library. However the functionality has now been rolled into the *libdsap*(3n) library and the *libquipu*(3n) library removed.

The library provides some routines to manage the data returned by the DUA procedure calls described in Chapter 17.

19.1 The Entry Structure

The `Entry` structure is used by the QUIPU DSA to store the data in the local DIT. This structure can also be used by a DUA to cache information from a successful read operation, and so the structure is described. In fact, this is how DISH maintains its cache.

The structure is shown below. Many of the fields are not appropriate to a DUA, but are briefly described for completeness.

```
typedef struct entry {
    RDN             e_name;
    Attr_Sequence   e_attributes;
    InheritAttr     e_iattr;
    char            e_leaf;
    char            e_complete;
    int             e_data;
#define E_DATA_MASTER            1
```

```
#define E_TYPE_SLAVE             2
#define E_TYPE_CACHE_FROM_MASTER  3
#define E_TYPE_CONSTRUCTOR       4
    char              e_allchildrenpresent;
    struct acl      * e_acl;
    DN                e_alias;
    struct dsa_info * e_dsainfo;
    char            * e_edbversion;
    AV_Sequence       e_oc;
    AV_Sequence       e_inherit;
    struct entry    * e_parent;
    Avlnode         * e_children;
    time_t            e_age;
    char              e_lock;
    char              e_external;
                      /* 0 -> Quipu, 1 -> External */
    union  {
       struct {
       AV_Sequence un_master;
       AV_Sequence un_slave;
       } un_in;
       struct {
       int         un_reftype;
       AV_Sequence  un_reference;
       } un_out;
    } e_un;
#define e_master        e_un.un_in.un_master
#define e_slave         e_un.un_in.un_slave
#define e_reference     e_un.un_out.un_reference
#define e_reftype       e_un.un_out.un_reftype

    int         e_refcount;
} entry, *Entry;
```

`e_name`: The RDN of the entry, to find the DN, use the routine

```
DN get_copy_dn (entryptr)
Entry entryptr;
```

`e_attributes`: The attributes returned by the DSA.

`e_iattr`: pointer to attributes inherited into the entry. This will always be NULL in a DUA — all attributes whether or not inherited will be in the `e_attributes` field.

`e_leaf`: Set to `TRUE` if the entry is known to be a leaf.

`e_complete`: Set to `TRUE` if it is know that we have *all* the attributes.

`e_data`: This takes one of four values. in the DUA only two apply. `E_TYPE_CACHE_FROM_MASTER` implies the entry has data that has been read from the DSA. `E_TYPE_CONSTRUCTOR` is used when the entry has been "made" by the caching mechanism to fill in a missing part of the DIT on the way down to another entry. An entry of this type will contain no data except from the RDN of the entry.

`e_allchildrenpresent`: Set to `TRUE` when it is known that all the children are held. This is a DSA-specific field.

`e_acl`: A pointer to the "Access Control List" attribute.

`e_alias`: Set to `TRUE` if the entry represents an alias. This is a DSA specific field.

`e_dsainfo`: pointers to the "EDBInfo" and "PresentationAddress" attribute of entries representing DSAs.

`e_edbversion`: This is a DSA specific field representing the EDB version number of the children.

`e_parent`: Pointer to the parent entry.

`e_children`: ACL pointer to the children.

`e_age`: If the data type is `E_TYPE_CACHE_FROM_MASTER`, then the entry is time stamped. This enables the cache to be "timed out". *Note*: in a DUA all entries are cached!

`e_lock`: This field has a dual purpose! In the DSA it is used to prevent an entry from being modified. In the DUA, if `FALSE`, then is suggests that in the field `e_attributes` there are only the attribute types for some attributes, this is a result of caching an results

from an operation in which the entry info selection requested that attribute type only were returned (see Section 17.7.1).

`e_external`: Indicates type of reference — QUIPU or non-QUIPU.

`e_master`: Pointer to the "MasterDSA" attribute.

`e_slave`: Pointer to the "SlaveDSA" attribute.

`e_reftype`: Type of a reference to a non-QUIPU DSA.

`e_reference`: Reference to a non-QUIPU DSA.

`e_refcount`: Reference count used to make sure entry is not freed too early.

Using this structure a tree, mapping the DIT can be built up. The tree is rooted by the external `Entry` "`database_root`". This entry will *never* have any attributes or siblings, *only* children.

19.2 Caching Results

There are four routines to create a cache from results of DAP operations:

To cache results returned from a `read` or `search` operation call

```
cache_entry (ptr, complete, vals)
EntryInfo      *ptr;
char            complete;
char            vals;
```

The `complete` parameter should be `TRUE` if all attributes were requested and returned.

The `vals` parameter should be `TRUE` if the attribute values were asked for.

The external `Entry` "`current_entry`" will be a pointer to the newly created entry.

To remove an entry from the cache use:

```
delete_cache (adn)
DN              adn;
```

19.3 Finding Data in the Cache

The procedure

```
Entry local_find_entry (object,deref)
DN                     object;
char                   deref;
```

is used to find an entry in the cache. If the entry exists, it will be returned, otherwise `NULLENTRY` will be returned. `object` is the DN of the entry you want to find. If `deref` is `TRUE` then any aliases encountered in trying to find the entry will be de-referenced.

The cache is only maintained for the period specified in a `cache_timeout` external variable (this is initially set to six hours). If the cache expires then `local_find_entry` will return `NULLENTRY`. To prevent timeouts, the routine `local_find_entry_aux` will return the cached entry even if the timeout has expired.

19.4 Caching List Results

List results are cached slightly differently, as only the RDN is known.

```
cache_list (ptr, prob,dn,sizelimit)
struct subordinate *ptr;
int             prob;
DN              dn;
int             sizelimit;
```

The parameters are as follows:

`ptr`: A pointer to the subordinates returned by the search.

`prob`: The `srr_limitproblem` field of the search results, see Section 17.10.

`dn`: The DN of the entry that has been listed.

`ptr`: The `svc_sizelimit` field of the common argument of the list argument (see Section 17.2.1).

To inspect the cache use

```
struct list_cache *find_list_cache (dn,sizelimit)
DN dn;
int sizelimit;
```

The `sizelimit` parameter is the sizelimit you would send in the list request.

This returns a `list_cache` structure; if the result is NULL, then the entry is not cached, or has less than "sizelimit" results.

```
struct list_cache {
        DN              list_dn;
        struct subordinate *list_subs;
        struct subordinate *list_sub_top;
        int             list_count;
        int             list_problem;
        struct list_cache *list_next;
};
```

The subordinates can be found in `list_sub_top`. The other field should be ignored, as they are for internal management only.

19.5 Changes

To conclude this part of the manual, a brief summary of the changes between this version and the QUIPU-6.0 version of the *libdsap*(3n) library is given.

Calls to the `x_decode()` routines are no longer required and should be removed. All the decoding is performed as the attributes come across the network.

The AttributeType structure is now just a pointer into the OID tables, thus "at_table" structure reference is no longer needed. Code of the form

```
at->at_table.oa_syntax
```

should be replaced with

```
at->oa_syntax
```

and

```
at->at_oid
```

replaced with

```
at->oa_at.ot_oid
```

Finally, to avoid a symbol clash, a search filter now uses the structure

```
s_filter
```

In previous releases of QUIPU this was just

```
filter
```

Part V

Appendices