`ContinuationRef` is returned as part of the error or results structures. The structure is explained below:

```
typedef struct continuation_ref {
    DN          cr_name;
    struct op_progress cr_progress;
    int         cr_rdn_resolved;
#define CR_RDNRESOLVED_NOTDEFINED  -1
    int         cr_aliasedRDNs;
#define CR_NOALIASEDRDNS -1
    int         cr_reftype;
#define RT_UNDEFINED    -1
#define RT_SUPERIOR     1
#define RT_SUBORDINATE  2
#define RT_CROSS        3
#define RT_NONSPECIFICSUBORDINATE 4
    struct access_point * cr_accesspoints;
    struct continuation_ref *cr_next;
}continuation_ref, *ContinuationRef;
```

`cr_name`: The DN that has only been partially explored.

`cr_rdn_resolved`: The number of RDNs in the name that have been examined; hence, how far down the DIT the query has been taken.

`cr_aliasedRDNs`: If `TRUE` then some of the RDNs were aliases.

`cr_reftype`: The type of reference that was used to get this information.

`cr_accesspoints`: The access point of the DSA to contact. The structure for an access point is as follows:

```
struct access_point {
    DN                  ap_name;
    struct PSAPaddr     *ap_address;
    struct access_point *ap_next;
};
```

`ap_name`: The DN of the DSA to contact.

`ap_address`: The address of the DSA (derivable from the name).

`ap_next`: There may be more than one access point, hence they form a linked list structure.

`cr_next`: There may be more than one access point; if so they form a linked list structure.

17.4 Errors

All DAP operations return a common error structure, which maps closely onto the service definitions. An error is only returned if the operation failed; if successful (indicated by the return value DS_OK) the error structure returned is undefined.

```
struct DSError {
    int dse_type;
#define DSE_INTR_ABANDON_FAILED -5
#define DSE_INTR_ABANDONED      -4
#define DSE_INTRERROR           -3
#define DSE_LOCALERROR          -2
#define DSE_REMOTEERROR         -1
#define DSE_NOERROR             0
#define DSE_ATTRIBUTEERROR      1
#define DSE_NAMEERROR           2
#define DSE_SERVICEERROR        3
#define DSE_REFERRAL            4
#define DSE_ABANDONED           5
#define DSE_SECURITYERROR       6
#define DSE_ABANDON_FAILED      7
#define DSE_UPDATEERROR         8
#define DSE_DSAREFERRAL         9
    union {
       struct DSE_attribute    dse_un_attribute;
       struct DSE_name         dse_un_name;
       struct DSE_service      dse_un_service;
       struct DSE_referral     dse_un_referral;
       struct DSE_abandon_fail dse_un_abandon_fail;
       struct DSE_security     dse_un_security;
       struct DSE_update       dse_un_update;
```

```
    }   dse_un;
};
```

The field `dse_type` is used to indicate what sort of error has occurred, and hence which structure from the union is used.

The value `DSE_LOCALERROR` is used to indicate that the error came from within the DUA; there is no associated structure in the union for this error.

The value `DSE_REMOTEERROR` is used to indicate that an error occurred at the DSA end, and the request was rejected; again there is no associated structure in the union for this error.

The structures in the union for the other error conditions are as described in the following sections.

17.4.1 Attribute Error

```
struct DSE_attribute {
    DN DSE_at_name;
    struct DSE_at_problem DSE_at_plist;
};
```

`DSE_at_name`: The name of the entry causing the error.

`DSE_at_plist`: A list of the errors:

```
struct DSE_at_problem {
    int                     DSE_at_what;
#define DSE_AT_NOSUCHATTRIBUTE          1
#define DSE_AT_INVALIDATTRIBUTESYNTAX   2
#define DSE_AT_UNDEFINEDATTRIBUTETYPE   3
#define DSE_AT_INAPPROPRIATEMATCHING    4
#define DSE_AT_CONSTRAINTVIOLATION      5
#define DSE_AT_TYPEORVALUEEXISTS        6
    AttributeType           DSE_at_type;
    AttributeValue          DSE_at_value;
    struct DSE_at_problem * dse_at_next;
};
#define DSE_AT_NOPROBLEM ((struct DSE_at_problem*)0)
```

The fields are used as follows:

`DSE_at_what`: Indicates which error has occurred.

`DSE_at_type`: The attribute type causing the error.

`DSE_at_value`: The associated value (if any).

`dse_at_next`: There may be more than one such error; if so, they form a linked list.

17.4.2 Name Error

```
struct DSE_name {
    int DSE_na_problem;
#define DSE_NA_NOSUCHOBJECT             1
#define DSE_NA_ALIASPROBLEM             2
#define DSE_NA_INVALIDATTRIBIBUTESYNTAX 3
#define DSE_NA_ALIASDEREFERENCE         4
    DN  DSE_na_matched;
};
```

`DSE_na_matched` is the part of the DN successfully matched.

17.4.3 Referral Errors

```
struct DSE_referral {
    ContininuationRef DSE_ref_candidates;
    DN                DSE_ref_prefix;
};
```

The continuation reference supplies information on how to continue the query. The structure is described in Section 17.3

The field `DSE_ref_prefix` is for DSP only. You should generally chase such referrals.

17.4.4 Security Error

```
struct DSE_security {
    int DSE_sc_problem;
#define DSE_SC_AUTHENTICATION           1
#define DSE_SC_INVALIDCREDENTIALS       2
```

```
#define DSE_SC_ACCESSRIGHTS             3
#define DSE_SC_INVALIDSIGNATURE         4
#define DSE_SC_PROTECTIONREQUIRED       5
#define DSE_SC_NOINFORMATION            6
};
```

17.4.5 Service Error

```
struct DSE_service {
    int DSE_sv_problem;
#define DSE_SV_BUSY                          1
#define DSE_SV_UNAVAILABLE                   2
#define DSE_SV_UNWILLINGTOPERFORM            3
#define DSE_SV_CHAININGREQUIRED              4
#define DSE_SV_UNABLETOPROCEED               5
#define DSE_SV_INVALIDREFERENCE              6
#define DSE_SV_TIMELIMITEXCEEDED             7
#define DSE_SV_ADMINLIMITEXCEEDED            8
#define DSE_SV_LOOPDETECT                    9
#define DSE_SV_UNAVAILABLECRITICALEXTENSION 10
#define DSE_SV_OUTOFSCOPE                   11
#define DSE_SV_DITERROR                     12
};
```

17.4.6 Update Error

```
struct DSE_update {
    int DSE_up_problem;
#define DSE_UP_NAMINGVIOLATION          1
#define DSE_UP_OBJECTCLASSVIOLATION     2
#define DSE_UP_NOTONNONLEAF             3
#define DSE_UP_NOTONRDN                 4
#define DSE_UP_ALREADYEXISTS            5
#define DSE_UP_AFFECTSMULTIPLEDSAS      6
#define DSE_UP_NOOBJECTCLASSMODS        7
};
```

17.4.7 Abandon Failure

If an abandon operation fails then the following is used:

```
struct DSE_abandon_fail {
    int DSE_ab_problem;
#define DSE_AB_NOSUCHOPERATION  1
#define DSE_AB_TOOLATE          2
#define DSE_AB_CANNOTABANDON    3
    int DSE_ab_invokeid;
};
```

17.4.8 Error Handling Procedures

The *libdsap*(3n) library has three routines for handling errors:

```
ds_error (ps,error)
PS ps;
struct DSError * error;
```

This routine will take the error, and pretty-print the contents to the PStream `ps`.

```
log_ds_error (error)
struct DSError * error;
```

This routine will take an error structure, print a simple message in *dsap.log* at the LLOG_EXCEPTIONS logging level based on the error, and a more detailed report at LLOG_TRACE logging level.

```
ds_error_free (err)
struct DSError * err;
```

This frees the embedded structures within the error structure, *but not* `DSError` itself.

17.5 Binding and Unbinding

Before operations may be invoked, the DUA must BIND to the DSA.

The `bind` procedure

```
secure_ds_bind (arg, error, result)
    struct ds_bind_arg          * arg;
    struct ds_bind_arg          * result;
    struct ds_bind_error        * error;
```

is used for this purpose.

You will need to include *quipu/bind.h* to use this procedure.

The `ds_bind_arg` structure is defined as follows.

```
struct ds_bind_arg {
    int  dba_version;
#define DBA_VERSION_V1988 0
    int dba_auth_type;
#define DBA_AUTH_NONE      0
#define DBA_AUTH_SIMPLE    1
#define DBA_AUTH_STRONG    2
#define DBA_AUTH_EXTERNAL  3
#define DBA_AUTH_PROTECTED 4
    char  *dba_time1;
    char  *dba_time2;
    struct random_number dba_r1;
    struct random_number dba_r2;
    DN     dba_dn;
    int    dba_passwd_len;
#define DBA_MAX_PASSWD_LEN 512
    char   dba_passwd[DBA_MAX_PASSWD_LEN];
    struct signature        *dba_sig;
    struct certificate_list *dba_cpath;
    char * dba_vtmp;
    int    dba_vlen;
};
```

`dba_version`: should always be set to `DBA_VERSION_V1988`.

`dsa_auth_type`: is used to select the type of authentication required.

`dba_dn`: is the name of the entity you wish to bind as, this can be `NULLDN` if you only require access to "publically readable" information.

`dba_passwd`: The password required to bind as the entity. This will be checked against the "userPassword" attribute in the entity.

`dba_passwd_len`: The length of the string in `dsa_passwd`.

`dba_sig`: The strong authentication signature (which must be provided if strong authentication is used in the bind call).

`dba_cpath`: The strong authentication certification path (optional).

`dba_time1`: Used for protected and strong authentication.

`dba_time2`: Optionally used for protected simple authentication.

`dba_r1`: Random number used for protected and strong authentication.

`dba_r2`: Optionally used for protected simple authentication.

`dba_vtmp`: Unused.

`dba_vlen`: Unused.

The `bind` operation will try to connect to the DSA, whose address is defined in the external "`char *`". "`dsa_address`", the syntax of the string is that of an ISODE PSAP address.

If `secure_ds_bind()` returns `DS_OK`, then you have a successful connection, otherwise a `ds_bind_error` structure will inform you of the error condition.

```
struct ds_bind_error {
    int  dbe_version;
    char dbe_type;
#define DBE_TYPE_SERVICE  1
#define DBE_TYPE_SECURITY 2
    int  dbe_value;
};
```

The filed `dbe_value` takes a value as defined for a `DSE_security` or `DSE_service` error shown in Sections 17.4.4 and 17.4.5.

17.5.1 No Authentication

To bind without using any authentication, `dsa_auth_type` should be set to `DBA_AUTH_NONE`. The fields `dba_version` and `dba_dn` must be filled in.

17.5.2 Simple Authentication

To bind using simple authentication, the `dsa_auth_type` field should be set to the value `DBA_AUTH_SIMPLE`. The `dba_version`, `dba_dn`, `dba_passwd` and `dba_passwd_len` fields must be filled in.

For backwards compatibility the routine

```
ds_bind (arg, error, result)
    struct ds_bind_arg          * arg;
    struct ds_bind_arg          * result;
    struct ds_bind_error        * error;
```

is still supplied, and gives you an unprotected simple bind request.

17.5.3 Protected Simple Authentication

To bind using protected simple authentication, the `dsa_auth_type` field should be set to to the value `DBA_AUTH_PROTECTED`. The fields `dba_version`, `dba_dn`, `dba_time1`, `dba_r1`, `dba_passwd` and `dba_passwd_len` must be filled in. The fields `dba_time2` and `dba_r2` must be either filled in or set to `NULL`.

17.5.4 Strong Authentication

To bind using strong authentication, the `dsa_auth_type` field should be set to the value `DBA_AUTH_STRONG`. The fields `dba_version`, `dba_dn`, `dba_time1`, `dba_r1`, and `dba_sig` must be filled in. The field `dba_cpath` must be either filled in or set to `NULL`.

17.6 Unbind

The unbind operation has no parameters.

```
ds_unbind ()
```

and is used to disconnect from the directory.

17.7 Read

The read operation is used to access information from a particular entity in the DSA.

```
ds_read (arg, error, result)
    struct ds_read_arg          * arg;
    struct ds_read_result       * result;
    struct DSError              * error;
```

You will need to include *quipu/read.h* to use this procedure.
The argument `ds_read_arg` is used to formulate the DAP request:

```
struct ds_read_arg {
    CommonArgs         rda_common;
    DN                 rda_object;
    EntryInfoSelection rda_eis;
};
```

The parameters are used as follows:

`rda_common`: The common arguments as described in Section 17.2.1.

`rda_object`: The DN of the object you want to read.

`rda_eis`: The "Entry Information Selection", which defines which attributes you want to be returned, this is defined in Section 17.7.1.

The results returned on a `DS_OK` return form the structure shown below:

```
struct ds_read_result {
    CommonResults rdr_common;
    EntryInfo     rdr_entry;
};
```

`rdr_common`: The common results as described in Section 17.2.2.

`rdr_entry`: This is a pointer to the "Entry Information" which contains the attributes you requested. This structure is defined in Section 17.7.2.