```
struct DAPindication {
    int     di_type;
#define DI_RESULT       2
#define DI_ERROR        3
#define DI_PREJECT      4
#define DI_ABORT        6
    union {
        struct DAPresult        di_un_result;
        struct DAPerror         di_un_error;
        struct DAPpreject       di_un_preject;
        struct DAPabort         di_un_abort;
    } di_un;
#define di_result di_un.di_un_result
#define di_error di_un.di_un_error
#define di_preject di_un.di_un_preject
#define di_abort di_un.di_un_abort
};
```

This structure is a discriminated union (tag element followed by a union). Depending on the routine called, style of behaviour requested, and the value returned by the routine the contents of the indication should be readily retrievable.

For the bind and unbind routines the `DAPindication` structure is only updated on failure and will contain an abort indication.

For the DAP remote operations the `DAPindication` structure is used to indicate failure of the association, failure of the request, an error response to a request, or a result response to a request.

When the `DAPindication` structure is used to indicate a failure of the association it will contain a `DAPabort` structure.

```
struct DAPabort {
    int     da_source;

    int     da_reason;

#define DA_SIZE 512
    int     da_cc;
    char    da_data[DA_SIZE];
}
```

`da_source`: the source of the abort, one of:

Value	Source
DA_USER	service-user (peer)
DA_PROVIDER	service-provider
DA_LOCAL	local DAPM

`da_reason`: The reason for aborting the association (if known).

`da_data/da_cc`: a diagnostic string from the provider.

When the `DAPindication` structure is used to indicate a rejection it will contain a `DAPpreject` structure.

```
struct DAPpreject {
    int           dp_id;

    int           dp_source;

    int           dp_reason;

#define DP_SIZE 512
    int     dp_cc;
    char    dp_data[DP_SIZE];
}
```

`dp_id`: the operation identifier of the operation rejected.

`dp_source`: the source of the abort, taking the same values as `da_source` in the `DAPabort` structure described above.

`dp_reason`: The reason for rejecting the operation (if known).

`dp_data/dp_cc`: a diagnostic string from the provider.

When the `DAPindication` structure is used to indicate a result in response to a previously issued remote operation request it will contain a `DAPresult` structure.

```
struct DAPresult {
    int                 dr_id;
    struct DSResult     dr_res;
}
```

`dr_id`: the operation identifier of the operation for which this is the result.

`dr_res`: the decoded result structure.

The operation identifier should be used on the receipt of an operation result to determine the operation the result applies to, and the result structure handled appropriately.

When the `DAPindication` structure is used to indicate a error in response to a previously issued remote operation request it will contain a `DAPerror` structure.

```
struct DAPerror {
    int                 de_id;
    struct DSError      de_err;
}
```

`de_id`: the operation identifier of the operation for which this is the error response.

`de_err`: the decoded error structure.

The operation identifier should be used on the receipt of an operation error to determine the operation the error applies to, and the error structure handled appropriately.

18.1.4 Return Values

The values returned by the various routines in this interface and their meanings are more complicated than may be entirely reasonable. However, a quick overview goes as follows:

- for bind and unbind operations synchronous style and DAP remote operations in synchronous or interruptible style, the request routines may return `NOTOK` on failure and `OK` on success.

- for bind and unbind operations asynchronous style, the request routines may return `NOTOK` on failure, `DONE` on completion, and `CONNECTING_1` or `CONNECTING_2` when incomplete.
- for DAP remote operations asynchronous style, the request routines may return `NOTOK` on failure, `OK` on successful invocation.

18.2 Binding and Unbinding

18.2.1 Binding

To be able to send DAP remote operations to a DSA it is necessary to establish an association with the DSA using a DAP bind operation. This can be accomplished either synchronously or asynchronously using the routine `DapAsynBindReqAux`.

```
int       DapAsynBindReqAux (callingtitle, calledtitle,
                callingaddr, calledaddr, prequirements,
                srequirements, isn, settings, sf,
                bindarg, qos, dc, di, async)
AEI                       callingtitle;
AEI                       calledtitle;
struct PSAPaddr         * callingaddr;
struct PSAPaddr         * calledaddr;
int                       prequirements;
int                       srequirements;
long                      isn;
int                       settings;
struct SSAPref          * sf;
struct ds_bind_arg      * bindarg;
struct QOStype          * qos;
struct DAPconnect       * dc;
struct DAPindication    * di;
int                       async;
```

This routine will attempt to call a DSA at the called address and/or with the called title, to establish a directory access association using the directory bind argument provided.

Often, all that is required for many of these parameters is that they are initialised to appropriate values. The routine `DapAsynBindRequest` is provided which sets up sensible defaults for most of the parameters before calling the routine `DapAsynBindReqAux`.

```
int DapAsynBindRequest (calledaddr, bindarg, dc, di, async)
struct PSAPaddr         * calledaddr;
struct ds_bind_arg      * bindarg;
struct DAPconnect       * dc;
struct DAPindication    * di;
int                       async;
```

`calledaddr:` the address of the DSA to send the bind request to. The `PSAPaddr` structure is described in *Volume Two*, and more general discussion of addresses is given in *Volume One*.

`bindarg:` is a representation of the argument of the DAP bind operation. The `ds_bind_arg` structure is described in Section 17.1.

`dc:` is used to return bind response information when a response is received. The `DAPconnect` structure comprises connection information from the underlying association and a structure containing a representation of the bind response (bind result or bind error) if any.

`di:` is the indication parameter described above in Section 18.1.3.

`async:` can take one of two values: `ROS_ASYNC` or `ROS_SYNC`. If the value is `ROS_SYNC`, then the routine will await a response to the association request and attempt to decode the value returned in that response. If the value is `ROS_ASYNC`, then the routine may return without receiving an association response, which may be checked for later using the `DapAsynBindRetry` routine.

The `DAPconnect` structure is used to return the response to a DAP bind operation:

```
struct DAPconnect {
    int     dc_sd;
```

```
    int     dc_pctx_id;
    struct AcSAPconnect dc_connect;

    int     dc_result;
#define DC_RESULT       1
#define DC_ERROR        2
#define DC_REJECT       3

    union {
        struct ds_bind_arg        dc_bind_res;
        struct ds_bind_error      dc_bind_err;
    } dc_un;
};
```

`dc_sd`: the association identifier assigned to the bound association.

`dc_pctx_id`: the identifier of the directory access context in the negotiated list of presentation contexts.

`dc_connect`: the connect information for the underlying association.

`dc_result`: the type of response received.

`dc_bind_res`: the decoded bind result if the response is a result.

`dc_bind_err`: the decoded bind error if the response is an error.

For explanations of the other parameters to `DapAsynBindReqAux` see the description of the `AcAssocRequest` routine on page 32 of *Volume One*.

When the asynchronous style of behaviour is selected for a bind request, the `DapAsynBindRetry` can be used to attempt to complete the bind and process any response. In order to determine when to call `DapAsynBindRetry`, the methods for asynchronously establishing connections described in Section 4.2.3 on page 112 in *Volume Two* should be applied. Essentially, the values `CONNECTING_1` and `CONNECTING_2`, are used to determine whether the association establishment is currently blocked on reading or writing, and is used in constructing a call to `xselect` to determine if further progress can be made. If the call to `xselect` indicates that further work can be done then `DapAsynBindRetry` should be called.

```
int       DapAsynBindRetry (sd, do_next_nsap, dc, di)
int                         sd;
int                         do_next_nsap;
struct DAPconnect         * dc;
struct DAPindication      * di;
```

`sd:` the association descriptor returned in the `DAPconnect` structure after an asynchronous call to `DapAsynBindReqAux`.

`do_next_nsap:` is used to specify whether the association should be retried on the same NSAP of the called address originally passed to `DapAsynBindReqAux` or whether to give up on the current NSAP and go on to the next (if any). A value of zero will specify retrying on the same NSAP, a non-zero value specifies retrying on the next NSAP. This is somewhat messy as it forces the calling code to keep track of how long a particular NSAP has been tried for and when to try another. If there are no more NSAPs when the next NSAP is requested, then the association attempt is deemed to have failed and an appropriate indication is generated.

`dc:` if the routine returns `DONE` then this structure will contain the connect information as described for `DapAsynBindReqAux` above.

`di:` if the routine returns `NOTOK` then this structure will contain the indication information.

18.2.2 Unbinding

When a DUA no longer needs to be bound to a DSA it can issue a DAP unbind operation to unbind from the DSA and end the association. This is achieved by using the `DapUnBindRequest` routine.

```
int       DapUnBindRequest (sd, secs, dr, di)
int                         sd;
int                         secs;
struct DAPrelease         * dr;
struct DAPindication      * di;
```

`sd:` the association identifier for the association from which the DUA wishes to unbind.

`secs`: the number of seconds to spend trying to unbind.

`dr`: the result of unbinding when complete.

`di`: an indication.

If the unbind operation fails then the routine will return `NOTOK` and an appropriate indication.

If the unbind operation completes successfully within `secs` seconds then the routine will return `OK` and the `DAPrelease` structure will be filled out.

If no unbind response is received within `secs` seconds the routine will return a value of `DONE` and the unbind should be completed using the routine `DapUnBindRetry`.

```
int       DapUnBindRetry (sd, secs, dr, di)
int                       sd;
int                       secs;
struct DAPrelease       * dr;
struct DAPindication    * di;
```

Which takes exactly the same parameters and has similar behaviour to the routine `DapUnBindRequest` above.

18.3 DAP Remote Operations

For each DAP remote operation (read, abandon, add entry, ...) there is a routine in the asynchronous interface to request that operation. The `DapRead` routine will be described in detail, which taken in conjunction with the descriptions of arguments given in Section 17 should also provide adequate description of the routines `DapCompare`, `DapAbandon`, `DapList`, `DapSearch`, `DapAddEntry`, `DapRemoveEntry`, `DapModifyEntry`, and `DapModifyRDN`.

All these routines can be invoked with synchronous, interruptible or asynchronous styles of behaviour. In the synchronous and interruptible cases the call will not return until a response is received (in the interruptible style an interruption will cause an automatic sending of an abandon request for the operation requested by the routine and will await and return the response to that operation). In the asynchronous case the routine will return after the request has been sent and specific procedures for receiving responses must be undertaken using the `DapInitWaitRequest` routine.

18.3.1 Invoking Requests

```
int       DapRead (ad, id, arg, di, asyn)
int                       ad;
int                       id;
struct ds_read_arg      * arg;
struct DAPindication    * di;
int                       asyn;
```

`ad:` the association identifier for the bound association over which to send the read operation.

`id:` the operation identifier to assign the the read operation.

`arg:` the read argument.

`di:` an indication parameter, only used if the request fails.

`asyn:` the behaviour style selector.

If `DapRead` is invoked with synchronous behaviour, then it will block until a response arrives, and hope that the response is for the operation just requested, although no guarantees are made that this is so.

If the interruptible style is selected then the call may be interrupted by some signals and an abandon operation for the operation requested is generated when this occurs, with the routine then awaiting both the response to the abandon operation and to the original operation before returning.

If the asynchronous style is selected, then the call issues the request and returns, the response should be expected in a subsequent call to the routine `DapInitWaitRequest`.

18.3.2 Receiving Responses

```
int       DapInitWaitRequest (sd, secs, di)
int                       sd;
int                       secs;
struct DAPindication    * di;
```

`sd:` the association identifier.

`secs`: the number of seconds to spend waiting for a response; `OK` will produce a polling effect and `NOTOK` will produce a blocking effect.

`di`: the indication structure.

If the `DapInitWaitRequest` routine returns `OK` then there is a response contained in the indication parameter. If it returns `DONE` then there was no response available within the specified time.

18.4 Programming Comments

This interface has been written with expectations of the following sort of program structure in mind.

Through interaction with a user or from arguments or tailoring, a DUA will undertake to attempt one or more DAP bind operations to one or more DSAs.

An asynchronous call to `DapAsynBindReqAux` should be made using an appropriate bind argument for each DSA that the DUA wishes to bind to. If the association completes then operations can be constructed and invoked on the bound association identified by the value returned in the `dc_sd` fields of the `DAPconnect` structure `dc_sd`.

If the value returned by the `DapAsynBindRequest` procedure call is either `CONNECTING_1` or `CONNECTING_2` then this value should be recorded along with the value returned in the `dc_sd` field of the `dc` parameter, which identifies the association even though it is not yet complete. Operations *must not* be invoked using this identifier until the bound association is completely established.

At a later stage, for each outstanding association request, depending on whether the last value returned for the association attempt on that association identifier is `CONNECTING_1` or `CONNECTING_2`, the routine `xselect` should be used to check for writing or reading on the value returned by `PSelectMask` for the given association identifier.

If the call to `xselect` indicates that writing (or reading) is available then the routine `DapAsynBindRetry` should be called for that association identifier.

This should be repeated until the association attempt fails or is completed, at which point operations can be constructed and invoked over the