The drawings contained in this Recommendation have been done in AUTOCAD

18.6.5 *Find naming context procedure*

18.6.5.1 *Introduction*

Figure 8/X.518 shows this procedure in the form of a diagram. Below is a textual description. In this it is assumed that the current value of Operation Progress is always returned upon exit of the procedure.

18.6.5.2 *Arguments*

The procedure makes use of the following arguments:

- the target object name (the purported name);
- operation progress.

18.6.5.3 *Results*

There are two cases of successful outcome.

The first of these returns:

- a reference;
- operation progress (updated appropriately).

The second of these returns:

- an indication that a suitable naming context was found locally;
- operation progress (updated appropriately).

18.6.5.4 *Errors*

One of the following errors may be returned:

- **ServiceError (unableToProceed);**
- **ServiceError (invalidReference).**

18.6.5.5 *Procedure*

1) If **nameResolutionPhase** is set to **completed** on entry, attempt to match the purported name against the context prefixes of the superior naming contexts of all the locally held naming contexts. If a match is found, return all the appropriate locally held naming contexts. If no match is found, return an **invalidReference ServiceError**.

2) If **nameResolutionPhase** is not set to **completed**, attempt to match context prefixes against a sequence of one or more RDNs in the initial portion of the purported name. For a match to be found, all RDNs in a context prefix must be matched. The context prefixes used are those of Naming Contexts for which this DSA has administrative authority. In case of multiple matches the one with the maximum number of matched RDNs is chosen.

   If a match is found, execute (3).

   If a match is not found, execute (5).

3) If **nameResolutionPhase** is **notStarted**, execute (4). If the number of RDNs in the initial portion of the purported name, matched as described in (2) above, is greater or equal to the **nextRDNToBeResolved** component of **OperationProgress**, then execute (4), otherwise execute (9).

4) The **nextRDNToBeResolved** is set to the number of matched RDNs plus 1 and the **nameResolutionPhase** is set to **Proceeding**. The context is returned and this procedure terminated.

   As a performance enhancement, the DSA may optionally match the purported name against the cross references held by the DSA. If more RDNs are matched against a cross reference than against the locally held context prefixes, then execute step (7).

   Note - The Name Resolution procedure will in case of this outcome call the Local Name Resolution.

5) If no match was found, the value of the **nameResolutionPhase** is checked. If the **nameResolutionPhase** is **notStarted**, execute (6).

If the **nameResolutionPhase** is **proceeding** or **completed**, then execute (9).

6) Using Cross Reference context prefixes, attempt to match against a sequence of one or more RDNs in the initial portion of the purported name. In case of multiple matches, the one with the maximum number of matched RDNs is chosen.

7) If a match was found to a cross reference, set the **nextRDNToBeResolved** to the number of RDNs in the chosen cross reference. The cross reference is returned and this procedure is terminated.

8) If no match was found to a cross reference, determine if the DSA is a first level DSA. If not, it will have a superior reference. Return this and terminate the procedure.

   If the DSA is a first level DSA, set **nextRDNToBeResolved** to one, and **nameResolutionPhase** to **proceeding**. Return the root naming context and terminate the procedure.

9) Check the value of the **referenceType** component of the **ChainingArgument**. If a non-specific subordinate reference was used, or the request came from a DUA, execute (10); otherwise, return **ServiceError** with **invalidReference** problem and terminate the procedure.

10) Compare the initial portion of the purported name to the context prefixes (minus their last RDN) of the locally held naming contexts. This effectively is a comparison to some of the naming contexts of the immediate superior to this DSA.

    f there is no match, return **ServiceError** with **invalidReference** problem and terminate the procedure.

    f a match is found, and the number of RDNs matched is less than in **nextRDNToBeResolved - 1**, return **ServiceError** with **invalidReference** problem; otherwise, return **ServiceError unableToProceed** problem. Terminate the procedure.

18.6.6 *Local Name Resolution*

18.6.6.1 *Introduction*

The Local Name Resolution matches RDNs in the purported name against internal knowledge references. It returns Found, Remote Reference, Alias Dereferenced, or Error indication.

Figure 9/X.518 shows this procedure in the form of a diagram. Below is a textual description.

18.6.6.2 *Arguments*

The procedure makes use of the following arguments:

- internal reference to naming context (with pointer to the entry whose name is the same as the context prefix);
- the target object name (the purported name);
- operation progress;
- the value of the **dontDereferenceAliases** service control;
- the value of the **aliasedRDNs** parameter;
- the value of the **aliasDereferenced** parameter.

18.6.6.3 *Results*

There are three cases of successful outcome.

The first of these returns:

- a reference;
- operation progress (updated appropriately).

The second of these returns:

- an indication that the entry was found locally;
- operation progress (updated appropriately).

The third of these returns:

- an indication that an alias was dereferenced;
- operation progress (set back to "not started").

18.6.6.4 *Errors*

One of the following errors may be returned:

- name error.

18.6.6.5 *Procedure*

The naming context returned by FindNaming Context will point to the entry of the root of the subtree. In the case of the root context, the entry is only a null entry.

1) f the internal reference is for an alias entry, execute step (7), otherwise step (2).
2) f all the RDNs in the purported name have been matched, then the target entry has been found. Set **nameResolutionPhase** to **completed**. An internal pointer is returned and the procedure terminated.

   Otherwise step (3) should be executed.

   Note - The matching could be attained with the context prefix on its own, or with the context prefix plus successive RDNs contained in internal references in the knowledge tree.
3) If an internal reference entry is found subordinate to the current entry in the knowledge tree which matches the next RDN in the purported name, then increment the **nextRDNToBeResolved**, set current entry to subordinate entry, and execute step (1) of this procedure again.
4) If the current entry has a subordinate reference whose RDN matches the next one in the purported name, return it and terminate the procedure.
5) If there are any non-specific subordinate references, subordinate to the current entry in the knowledge tree, return them as references and terminate the procedure.
6) If an internal reference, subordinate reference, or non-specific subordinate reference is not found, then check the number of RDNs in the purported name that have been matched. If more RDNs have been matched than in the **aliasedRDNs** component of **ChainingArgument**, then return **NameError** with **noSuchObject** problem. If less RDNs have been matched, then return **NameError** with **aliasProblem**.
7) If the number of RDNs in the purported name that have been matched is less than or equal to the **aliasedRDNs** component of **ChainingArgument** (if any), then the previous alias that was dereferenced (if any) points to another alias. If so, return **NameError** with **aliasDereferencingProblem**.
8) If the **aliasedRDNs** component is missing, or if the number of RDNs matched is greater than **aliasedRDNs** component of **ChainingArgument**, then check the **dontDereferenceAlias** service control. If aliases can be dereferenced, then execute step (9), otherwise step (10).
9) Dereference the alias. Set **nameResolutionPhase** of **OperationProgress** to **notStarted**. Set **aliasDereferenced** component of **ChainingArgument** to **TRUE**, and **aliasedRDNs** to the number of RDNs in the **aliasedObjectName** attribute of the alias entry. Set **targetObject** to the new name. Terminate the procedure. (The process of Name Resolution will be restarted.)
10) If all the RDNs in the purported name have been matched, execute step (2). Otherwise, return **NameError** with **aliasDereferencingProblem**.

18.7 *Object evaluation procedures*

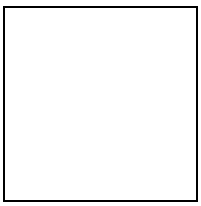
The object evaluation procedures specified comprise two categories of procedures:

a) single-object evaluation procedure;

b) multiple-object evaluation procedures.

Figure 10/X.518 shows the object evaluation procedure.

FIGURE 10/X.518 - T0704600-88



18.7.1 *Single-object evaluation procedures*

Single-object evaluation procedures, which are common to the class of operations concerned with accessing a single object are carried out directly, with the result or error being returned to the invoker.

These operations comprise **Read**, **Compare**, **AddEntry**, **RemoveEntry**, **ModifyEntry** and **ModifyRDN**, and their Chained counterparts.

The action required on the entry is as described in the appropriate paragraph of Recommendation X.511.

AddEntry, **RemoveEntry**, and **ModifyRDN** operations affect knowledge. If the immediate superior of the entry is in a different DSA, correct external knowledge references shall be maintained. How this is done is outside the scope of this Recommendation.

How the DSA is chosen to contain the entry created by **AddEntry** is outside the scope of this Recommendation.

If the immediate superior of an entry to be created by **AddEntry** or modified by **ModifyRDN** has non-specific subordinate references, procedures outside the scope of this Recommendation shall be followed to ensure that no two entries have the same distinguished name.

Requests which cannot be satisfied under these conditions shall fail with an **UpdateError** with problem **affectsMultipleDSAs**.

18.7.2 *Multiple-object evaluation procedures*

Multiple-object evaluation procedures, which are common to the class of operations concerned with accessing multiple objects, are specified in the following subparagraphs.

These operations comprise **List** and **Search**, and their Chained counterparts.

18.7.2.1 *List*

This paragraph specifies the evaluation procedure specific to **List** and **ChainedList**. (In what follows the term "List" applies to both.)

18.7.2.1.1 *List procedure* (I)

This procedure applies where the List request has n**ameResolutionPhase** component of **OperationProgress** set to **notStarted** or **proceeding** and where the DSA, after performing Name Resolution The base object will be denoted by "e".

1) Get each locally held immediate subordinate of e to form a local set of results. Set **aliasEntry** and **fromEntry** in **ListResult** as appropriate.

2) Get the set of non-specific subordinate references and subordinate references to DSAs which hold immediate subordinates of "e".

3) Pass the subrequest with base object = e, and **OperationProgress** set to **completed** to the Operation Dispatcher which subsequently forwards it to each DSA which holds immediate subordinates of e.

Note - If the DSA holds subordinate references with an indication of whether or not the subordinate entry are aliases, and the **dontUseCopy** is **FALSE**, then this step can be omitted for those entries. The information about the subordinates is available directly.

18.7.2.1.2 *List Procedure* (II)

This procedure applies to a List request with the **nameResolutionPhase** component of **OperationProgress** set to **completed**.

The base object will be denoted by "e".

1) Get each locally held immediate subordinate of e to form a local set of results. Set **aliasEntry** and **fromEntry** in **ListResult** as appropriate.

2) Pass the results to the Operation Dispatcher which forwards them to the requesting DUA or DSA.

18.7.2.2 *Search*

This paragraph specifies the evaluation procedure specific to **Search** and **ChainedSearch**. (In what follows the term "Search" applies to both.)

Note that two circumstances exist, requiring two separate procedures. The first procedure (18.7.2.2.1) applies when the DSA executing the Search contains the **targetObject** as a local entry. The second procedure (18.7.2.2.2) applies when the DSA executing the Search does not hold the **targetObject**, but only subordinates of the **targetObject**.

18.7.2.2.1 *Search procedure* (I)

This procedure applies to a Search request with the **nameResolutionPhase** component of **OperationProgress** set to **notStarted** or **proceeding** and where the DSA, after performing Name Resolution, determines that it holds the target object.

The base object will be denoted by "e".

1) If the **subset** argument is **baseObject** or **wholeSubtree**, then apply the filter argument specified in the Search to the entry e, to form a set of local results. Return the results for Results Merging. If the **subset** argument is **baseObject**, terminate the procedure, otherwise continue at (2).

2) If the **subset** argument is **oneLevel** or **wholeSubtree** form a set E from the locally-held immediate subordinates of e, except that:

   If aliases are to be dereferenced, i.e. the **searchAliases** parameter is **TRUE**, then any alias entries that are found are handled in paragraph 5) below and do not contribute to these results.

   Apply the filter arguments to E to give a filtered subset Ew'gE; return this set Ew' of local results for Results Merging.

3) Other subordinates of e may reside in other DSAs, and if so will be referenced as subordinate or non-specific subordinate references. For each DSA which is so referenced, prepare a new Search with targetObject = e, and with **nameResolutionPhase** of **OperationProgress** set to **completed**. Return each Search subrequest to the Operation Dispatcher for forwarding. If any error result is returned from a subrequest, it is ignored, as if no subrequest had been sent.

4) If the **subset** argument is **oneLevel**, the Search is now complete so terminate the procedure.

   If the **subset** argument is **wholeSubtree**, then:

   if the set E from paragraph (2) is empty, then the whole subtree held in this DSA has been searched, so terminate the procedure;

   otherwise continue processing as follows:

   let each entry that was in set E be denoted by e. Repeat the Search procedure from paragraph (2), for each entry e.

5) If aliases are to be dereferenced, any alias entries found in step (2) are placed in set D. For each entry d in D, dereference the alias, and formulate a new Search with **nameResolutionPhase** set to **notStarted**, and **targetObject** created from the **aliasedObjectName** attribute and the old **targetObject** name.

   If the **subset** argument was **oneLevel**, set it to **baseObject** in the new subrequest, otherwise set it to **wholeSubtree**.

   If any error result is returned from the subrequest, it is ignored, as if no subrequest had been made.

18.7.2.2.2 *Search Procedure* (II)

This procedure applies to a Search request with the **nameResolutionPhase** component of **OperationProgress** set to **completed**.

The target object will be denoted by "e".

For each locally held immediate subordinate ew' of e, formulate a new request with targetObject = ew'.

If the **subset** argument was **oneLevel,** set it to **baseObject,** otherwise leave it as **wholeSubtree**. Now carry out the procedure defined in steps (1) to (5) in 18.7.2.2.1. If there are no such subordinates, return **unableToProceed ServiceError**.

18.8 *Result merging procedure*

This procedure is called when external results and/or errors are present. There might also be one internal result. All results and errors are assumed to be held within the DSA until the procedure completes.

The external information could be due to chaining, multicasting or request decomposition.

In the case of chaining there will be a single result or error. In the case of multicasting there might be either no result, one result or several identical results. In addition, there may be some errors. If there is more than one result, all but one of them are arbitrarily discarded. A result is always returned in preference to an error. If there are no results, an error is returned, with the following exceptions:

i) If **invalidReference** was returned, the reference is marked as such, and the DSA may either use an appropriate alternate external reference to continue the request, or return **ditError** to the requestor. (The handling of invalid external references is beyond the scope of this Recommendation.)

ii) In the case of multicasting, **unableToProceed** errors should be ignored, unless all responses are of this type in which case **NameError noSuchObject** should be returned to the responder. If at least one result is returned, then all errors can be ignored.

iii) In the case of referrals, these need not be treated as errors, and may be acted upon.

If the merging is required due to a request decomposition, the merging amounts to forming the union of the results.

In the case of decomposition, when there are both results and errors to be merged, an incomplete result is returned to the requestor.

A DSA might at this stage choose to extract referrals from the incoming results and errors that should be merged. It might then decide to explore all or some of these further, in which case operations are chained. The old result will have to be saved and later merged with the results or errors produced by the chaining.

The handling of signatures which may be present with the results being returned is specified in 18.9.2 below.

18.9 *Procedures for distributed authentication*

This paragraph specifies the procedures necessary to support the directory distributed authentication services. These services, and hence the procedures, are categorized as:

- originator authentication, which is supported in either an unprotected (simple identity based) or secure (based upon digital signatures) form; and
- results authentication which is similarly protected (again based upon digital signatures).

18.9.1 *Originator authentication*

18.9.1.1 *Identity based authentication*

The identity based authentication service enables DSAs to authenticate the original requestor of information for the purpose of effecting local access controls. DSAs wishing to exploit this service must adopt the following procedure:

- for a DSA requiring to authenticate a DAP request, the DSA acquires the distinguished name of the requestor through the Bind procedures at the time a DUA association (DUA or DSA) is established. Successful conclusion of these procedures does not in any way prejudice the level of authentication that may subsequently be required for processing operations using that association;

- the DSA with which the DUA association exists must insert the requestor's distinguished name in the initiator field of the ChainingArgument for all subsequent chained operations to other DSAs;
- a DSA, on receiving a chained-operation, may satisfy that operation, or not, depending upon the determination of access rights (a locally defined mechanism). If the outcome is not satisfactory a **SecurityError** may be returned with **SecurityProblem** set to **insufficientAccessRights**.

18.9.1.2 *Signature-based originator authentication*

This signature-based originator authentication service enables a DSA to authenticate (in a secure manner) the originator of a particular service request. The procedures to be effected by a DSA in realizing this service are described in this paragraph.

The signature-based authentication service is invoked by a DUA using the **SIGNED** variant of an optionally-signed service request.

A DSA, on receiving a signed request from another DSA, shall remove that DSA's signature prior to processing the operation. Assuming the result of any signature verification proves to be satisfactory, the DSA will continue to progress the operation. If, during processing, the DSA requires to perform chaining, multicasting or request decomposition, the argument set for each associated chained operation shall be constructed as follows:

- the DSA forms an argument set which may be optionally signed; the argument set comprises the incoming signed argument set together with a modified **ChainingArgument**.

In the event that the DSA is able to contribute information to the response, originator authentication, based upon the signed service request, may be used for the determination of access rights to that information.

If a DSA receives an unsigned service request for information which will only be released subject to originator authentication, a **SecurityError** will be returned with **SecurityProblem** set to **protectionRequired**.

18.9.2 *Results authentication*

This service is provided to enable requestors of directory operations (either DUA or DSAs) to verify (in a secure manner using digital signature techniques) the source of results. The results authentication service may be requested irrespective of whether originator authentication is to be used.

The results authentication service is initiated using the **signed** value of the **protectionRequest** component as contained within the argument set of directory operations; a DSA receiving an operation with this option selected may then optionally sign any subsequent results. The **signed** option in the **protectionRequest** serves as an indication, to the DSA, of the requestor's preference; the DSA may, or may not, actually sign any subsequent results.

In the case where a DSA performs chaining, multicasting or request decomposition of such a request, the DSA has a number of options in terms of the form of results sent back to the requestor, namely:

a) return a composite response (signed or unsigned) to the requestor;

b) return a set of two or more uncollated partial responses (signed or unsigned) to the requestor; within this set zero or more members may be signed and zero or one unsigned. In the event that an unsigned partial result is present, this member may in fact be a collation of one or more unsigned partial responses which have been received from other DSAs, contributed by this DSA, or both.

ANNEX A

(to Recommendation X.518)

ASN.1 for distributed operations

This Annex is part of the Recommendation.

This Annex includes all of the ASN.1 type, value and macro definitions contained in this Recommendation in the form of the ASN.1 module **Distributed Operations**.

```
DistributedOperations {joint-iso-ccitt ds(5) modules(1) distributedOperations(3)}
DEFINITIONS ::=
BEGIN
EXPORTS
        DirectoryRefinement, chainedReadPort, chainedSearchPort, chainedModifyPort,
```

```
        DSABind, DSABindArgument,
        DSAUnbind,
        ChainedRead, ChainedCompare, ChainedAbandon,
        ChainedList, ChainedSearch,
        ChainedAddEntry, ChainedRemoveEntry,
        ChainedModifyEntry, ChainedModifyRDN,
        DsaReferral, ContinuationReference;

IMPORTS

        InformationFramework, abstractService, distributedOperations,
        directoryObjectIdentifiers, selectedAttributeTypes
            FROM    UsefulDefinitions {joint-iso-ccitt ds(5) modules(1)  usefulDefinitions(0)}

        DistinguishedName, Name, RelativeDistinguishedName
              FROM  InformationFramework informationFramework
        id-ot-dsa, id-pt-chained-read, id-pt-chained-search, id-pt-chained-modify
            FROM    DistributedDirectoryObjectIdentifiers, distributedDirectoryObjectIdentifiers

PresentationAddress
            FROM    SelectedAttributeTypes selectedAttributeTypes
        directory, readPort, searchPort, modifyPort
        DirectoryBind,
        ReadArgument, ReadResult,
        CompareArgument, CompareResult,
        Abandon
        ListArgument, ListResult,
        SearchArgument, SearchResult,
        AddEntryArgument, AddEntryResult,
        RemoveEntryArgument, RemoveEntryResult,
        ModifyEntryArgument, ModifyEntryResult,
        ModifyRDNArgument, ModifyRDNResult,
        Abandoned, AttributeError, NameError, ServiceError, SecurityError, UpdateError
        OPTIONALLY-SIGNED, SecurityParameters
            FROM    DirectoryAbstractService directoryAbstractService

-- objects and ports --

DirectoryRefinement    ::=  REFINE  directory AS
        dsa    RECURRING
            readPort            [S]     VISIBLE
            searchPort          [S]     VISIBLE
            modifyPort          [S]     VISIBLE
            chainedReadPort             PAIRED  WITH  dsa
            chainedSearchPort           PAIRED  WITH  dsa
            chainedModifyPort           PAIRED  WITH  dsa

dsa  OBJECT
        PORTS  {    readPort        [S],
                    searchPort      [S],
                    modifyPort      [S],
                    chainedReadPort,
                    chainedSearchPort
                    chainedModifyPort}
        ::=  id-ot-dsa

chainedReadPort PORT
        ABSTRACT OPERATIONS {
            ChainedRead, ChainedCompare,
            ChainedAbandon}
        ::=  id-pt-chained-read
```

```
chainedSearchPort PORT
        ABSTRACT OPERATIONS {
                ChainedList, ChainedSearch}
        ::=     id-pt-chained-search

chainedModifyPort PORT
        ABSTRACT-OPERATIONS {
                ChainedAddEntry, ChainedRemoveEntry,
                ChainedModifyEntry, ChainedModifyRDN}
        ::=  id-pt-chained-modify

DSABind     ::=         ABSTRACT-BIND
        TO  {chainedRead,
                chainedSearch,
                chainedModify}
        DirectoryBind

DSAUnbind   ::=         UNBIND
        FROM                {chainedRead,
                            chainedSearch,
                            chainedModify}

-- operations, arguments and results --

ChainedRead ::=
        ABSTRACT-OPERATION
                ARGUMENT    OPTIONALLY-SIGNED       SET{
                                                    ChainingArgument,
                                                    [0]  ReadArgument}
                RESULT      OPTIONALLY-SIGNED       SET{
                                                    ChainingResult,
                                                    [0]  ReadResult}
                ERRORS {
                        DsaReferral, Abandoned, AttributeError, NameError,
                        ServiceError, SecurityError}

ChainedCompare              ::=
        ABSTRACT-OPERATION
                ARGUMENT    OPTIONALLY-SIGNED       SET{
                                                    ChainingArgument,
                                                    [0]  CompareArgument}
                RESULT      OPTIONALLY-SIGNED       SET{
                                                    ChainingResult,
                                                    [0]  CompareResult}
                ERRORS {
                        DsaReferral, Abandoned, AttributeError, NameError,
                        ServiceError, SecurityError}

ChainedAbandon              ::=     Abandon

ChainedList  ::=
        ABSTRACT-OPERATION
                ARGUMENT    OPTIONALLY-SIGNED       SET{
                                                    ChainingArgument,
                                                    [0]  ListArgument}
                RESULT      OPTIONALLY-SIGNED       SET{
                                                    ChainingResult,
                                                    [0]  ListResult}
                ERRORS {
                        DsaReferral, Abandoned, AttributeError, NameError,
                        ServiceError, SecurityError }
```

```
ChainedSearch ::=
        ABSTRACT-OPERATION
                ARGUMENT    OPTIONALLY-SIGNED       SET{
                                                    ChainingArgument,
                                                    [0]  SearchArgument}
                RESULT      OPTIONALLY-SIGNED       SET{
                                                    ChainingResult,
                                                    [0]  SearchResult}
                ERRORS {
                        DsaReferral, Abandoned, AttributeError, NameError,
                        ServiceError, SecurityError}

ChainedAddEntry         ::=
        ABSTRACT-OPERATION
                ARGUMENT    OPTIONALLY-SIGNED       SET{
                                                    ChainingArgument,
                                                    [0]  AddEntryArgument}
                RESULT      OPTIONALLY-SIGNED       SET{
                                                    ChainingResult,
                                                    [0]  AddEntryResult}
                ERRORS {
                        DsaReferral, Abandoned, AttributeError, NameError,
                        ServiceError, SecurityError, UpdateError}

ChainedRemoveEntry   ::=
        ABSTRACT-OPERATION
                ARGUMENT    OPTIONALLY-SIGNED       SET{
                                                    ChainingArgument,
                                                    [0]  RemoveEntryArgument}
                RESULT      OPTIONALLY-SIGNED       SET{
                                                    ChainingResult,
                                                    [0]  RemoveEntryResult}
                ERRORS {
                        DsaReferral, Abandoned, NameError,
                        ServiceError, SecurityError, UpdateError}

ChainedModifyEntry    ::=
        ABSTRACT-OPERATION
                ARGUMENT    OPTIONALLY-SIGNED       SET{
                                                    ChainingArgument,
                                                    [0]  ModifyEntryArgument}
                RESULT      OPTIONALLY-SIGNED       SET{
                                                    ChainingResult,
                                                    [0]  ModifyEntryResult}
                ERRORS {
                        DsaReferral, Abandoned, AttributeError, NameError,
                        ServiceError, SecurityError, UpdateError}

ChainedModifyRDN      ::=
        ABSTRACT-OPERATION
                ARGUMENT    OPTIONALLY-SIGNED       SET{
                                                    ChainingArgument,
                                                    [0]  ModifyRDNArgument}
                RESULT      OPTIONALLY-SIGNED       SET{
                                                    ChainingResult,
                                                    [0]  ModifyRDNResult}
                ERRORS {
                        DsaReferral, Abandoned, NameError,
                        ServiceError, SecurityError, UpdateError}
```

```
-- errors and parameters --
DSAReferral ::=
        ABSTRACT-ERROR
                PARAMETER SET {
                        [0]     ContinuationReference,
                        contextPrefix   [1]     DistinguishedName OPTIONAL}

-- common arguments/results --
ChainingArguments       ::=             SET {
        originator                      [0]     DistinguishedName OPTIONAL,
        targetObject                    [1]     DistinguishedName OPTIONAL,
        operationProgress               [2]     OperationProgress DEFAULT {notStarted}
        traceInformation                [3]     TraceInformation,
        aliasDereferenced               [4]     BOOLEAN DEFAULT FALSE,
        aliasedRDNs                     [5]     INTEGER OPTIONAL,
                -- absent unless aliasDereferenced is TRUE
        returnCrossRefs                 [6]     BOOLEAN DEFAULT FALSE,
        referenceType                   [7]     ReferenceType DEFAULT superior,
        info                            [8]     Domaininfo OPTIONAL,
        timeLimit                       [9]     UTCTime OPTIONAL,
                                        [10]    SecurityParameters DEFAULT { }}

ChainingResults         ::=             SET {
        info                            [0]     Domaininfo OPTIONAL,
        crossReferences                 [1]     SEQUENCE OF CrossReference OPTIONAL,
                                        [2]     SecurityParameters DEFAULT { }}

CrossReference          ::=             SET {
        contextPrefix   [0]             DistinguishedName,
        accessPoint                     [1]     AccessPoint}

ReferenceType           ::=             ENUMERATED {
            superior                                    (1),
            subordinate                                 (2),
            cross                                       (3),
            nonSpecificSubordinate                      (4)}

TraceInformation        ::=             SEQUENCE OF
        SEQUENCE {
              targetObject              Name,
              dsa Name,
              OperationProgress}

OperationProgress       ::=             SET     {
        nameResolutionPhase                     [0]  ENUMERATED {
                                                        notStarted      (1),
                                                        proceeding      (2),
                                                        completed       (3)},
        nextRDNToBeResolved                     [1]  INTEGER OPTIONAL}

DomainInfo              ::=     ANY

ContinuationReference           ::=     SET {
        targetObject            [0]     Name,
        aliasedRDNs             [1]     INTEGER OPTIONAL,
        operationProgress       [2]     OperationProgress
        rdnsResolved            [3]     INTEGER OPTIONAL,
        referenceType           [4]     ReferenceType OPTIONAL,
                                -- only present in the DSP --
        accessPoints            [5]     SET OF AccessPoint }
```

```
AccessPoint  ::=     SET {
        ae-title     [0]    Name,
        address      [1]    PresentationAddress }
```

ANNEX B

(to Recommendation X.518)

Modelling of knowledge

This Annex is not part of the Recommendation.

B.1 *Example of knowledge modelling*

The following example illustrates the knowledge information that would have to be maintained by the DSAs shown in Figure 5/X.518 (9). Figure 5/X.518 depicts a hypothetical DIT logically partitioned into five Naming Contexts (A, B, C, D and E) and physically distributed over three DSAs (DSA1, DSA2, DSA3). In the example, DSA1 holds context C, DSA2 holds contexts A, B, and E, and DSA3 holds context D.

The following abbreviations have been used in Figures B-1/X.518 to B-3/X.518.

SUPR:	superior reference
SUBR:	subordinate reference
INTR:	internal reference
NSSR:	non-specific subordinate reference
CROSSR:	cross reference
DSAn:	Distinguished Name of DSAn
PS:	Presentation Address
CP:	context prefix
RDN:	Relative Distinguished name
DSA:	Distinguished name of a DSA
PTR:	Pointer
AON:	Aliased Object Name**.**

Note - The following figures are intended only to provide a pictorial example of the concepts defined in this paragraph. How knowledge information is actually stored and managed in a particular DSA implementation is a local matter and is outside the scope of this Recommendation.

FIGURE B-1/X.518 - T0704610-88

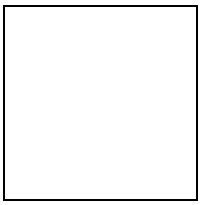


Figure B-1/X.518 illustrates the knowledge information that must be held by DSA1. This must include the following context prefixes and set of references:

Context Prefixes:	{C=WW, O=ABC}, context C.
Cross References:	{ }
Superior References:	{DSA2, presentation address of DSA2}
Internal References for Context C:	{C=WW, O=ABC},

{OU=G}, {OU=H}
{OU=G, CN=1},
{OU=G, CN=m},
{OU=G, CN=n}.

Subordinate References: { }

Non-specific subordinate References: {DSA2, presentation address of DSA2}.

FIGURE B-2/X.518 - T0704620-88

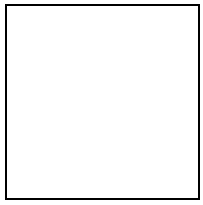


Figure B-2/X.518 illustrates the knowledge information that must be held by DSA2. This must include the following context prefixes and set of references:

Context Prefixes: {C=WW}, context A
{C=VV}, context B
{C=WW}, O=ABC, OU=I}, context E.

Cross References: { }

Superior References: { }

Internal References for Context A: {C=WW}

Internal References for Context B: {C=VV}

Internal References for Context E: {C=WW, O=ABC, OU=I},
{CN=o},
{CN=p},
{CN=q}.

Subordinate References for Context A: {C=WW, O=ABC}

Subordinate References for Context B: {C=VV, O=DEF}

Non-specific subordinate References: { }

FIGURE B-3/X.518 - T0704630-88

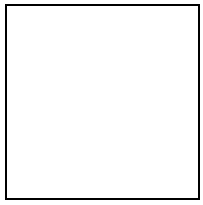


Figure B-3/X.518 illustrates the knowledge information that must be held by DSA3. This must include the following context prefixes and set of references:

Context Prefixes:	{C=VV, O=DEF}, context D
Cross References:	{{C=WW, O=ABC, OU=H}, DSA1, presentation address of DSA1} (not shown in the figure above)
Superior References:	{DSA2, presentation address of DSA2}
Internal References for Context D:	{DSA1, presentation address of DSA1} {C=VV, O=DEF}, {OU=J}, {OU=K} alias for {C=WW, O=ABC, OU=I, CN=o} (alias information is not part of the knowledge)
Subordinate References:	{ }
Non-specific subordinate References:	{ }

B.2 *Example of distributed name resolution*

The following is an example of how Distributed Name Resolution is used to process different directory requests. The example is based on the hypothetical DIT shown in Figure 5/X.518 (9) and the corresponding DSA configuration(s) shown in Figures B-1/X.518 to B-3/X.518 (Annex B).

Assuming a chaining mode of propagating, the following requests addressed to DSA1 would be processed as follows:

1) A request with distinguished name {C=WW, O=ABC, OU=G, CN=1}
   - Will match context prefix {C=WW, O=ABC} of context C for which DSA1 has administrative authority. Therefore, name resolution will begin in DSA1 with context C.
   - Name resolution will proceed downwards in context C successfully matching each remaining RDN, until CN=1 is located.
2) A request with distinguished name {C=WW, O=JPR}
   - Will not match any context prefix held by DSA1, therefore DSA1 will use its superior reference to forward the request to its superior DSA, DSA2.
   - In DSA2, the request will match context prefix {C=WW} and name resolution will begin in DSA2 with context A.
   - Name resolution will not find a subordinate of C=WW to match RDN O=JPR, therefore the request will fail and the name will be determined to have been invalid (i.e. reference a non-existent object).
3) A request with distinguished name {C=VV, O=DEF, OU=K}
   - DSA1 will therefore forward the request to its superior DSA, DSA2.
   - The request will match context prefix {C=VV} of context B held by DSA2. Therefore, name resolution will begin in DSA2 with context B.
   - As name resolution attempts to match O=DEF, it will find a subordinate reference indicating that {C=VV, O=DEF} is the start of a new context held in DSA3.
   - Name resolution will continue in DSA3 until {C=VV, O=DEF, CN=K} is located.
   - Assuming that aliases are to be dereferenced, a new name will be constructed using the aliased name contained in the entry {C=VV, O=DEF, CN=K}. The resulting new name will be: {C=WW, O=ABC, OU=I, CN=o}.
   - DSA3 will resume processing of the request using the new name obtained by dereferencing.

ANNEX C

(to Recommendation X.518)

Distributed use of authentication

This Annex is not part of the Recommendation.

C.1 *Summary*

The security model is defined in 10 of Recommendation X.501. The following is a summary of the main points of the model.

a) Simple Authentication of the operation initiator is not supported in the DSP.

b) Strong Authentication, by the signing of the request and of the result, is supported in the DSP.

c) Encryption of the request, or of the result, is not supported in the DSP.

d) Authentication of errors, including referrals, is not supported in the DSP.

This Annex describes how b) above is realized in the distributed Directory. It makes use of terminology and notation defined in Recommendation X.509.

C.2 *Simple authentication*

The DUA will be authenticated as part of the Bind Operation of the DAP. Thereafter, only the name of the DUA will be carried in the DSP, in the initiator field of the Chaining Argument.

C.3 *Distributed authentication model*

FIGURE C-1/X.518 - T0704640-88

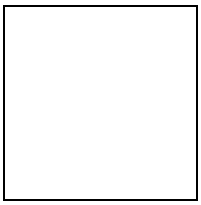


Figure C-1/X.518 illustrates the model to be used to specify the distributed authentication procedures. The model identifies the sequence of information flows for the general case of a list or search operation. The operation is considered as originating from DUA "a" citing a target object which resides in DSA "c"; in performing the operation, DSAs "b", "c", "d" and "e" are to be involved.

DUA "a" initially contacts any DSA (DSA "b") which does not hold the target object, but which is able to navigate, via chaining, to the DSA (DSA "c") holding the target object. If all the DSAs were operating in referral mode, then the model would be significantly simplified, and each DUA/DSA exchange would equate, in authentication terms, to the interaction between DUA "a" and DSA "b".

C.4 *DUA to DSA*

Originator authentication is realized as a consequence of exchange (1). In Figure C-1/X.518 the authentication procedure is as follows:

Let
OA = the Operation Argument i.e. Search, Read, Compare etc. Argument as defined in Part 3.

and

a(OA) = the Operation Argument signed by DUA "a".
Authentication will be determined by verification of the signature.

C.5 *Transference from the DAP to the DSP*

This procedure is effected by DSA "b" in Figure C-1/X.518 and represents the transference of the signed identity of the initiator from the DAP to the DSP.

DSA "b" formulates the appropriate Chaining Argument as described in 12.3 of this Recommendation and combines it with the Operation Argument from the DAP thus forming a Chained Operation, i.e. Chained Read, Search, List etc. of the DSP. The Chained Operation so formed will be signed prior to passing it to other DSAs (DSA "c" in Figure C-1/X.518). The data structure can be represented as:

b{ChA,a{OA}} = the Chained Operation signed by
DSA b

where

ChA = Chaining Argument.

Authentication information carried in the DSP between two DSAs [labelled exchange (2) in Figure C-1/X.518] therefore comprises two parts:

- the Operation Argument, signed by the initiator, which allows authentication of the initiator;
- the Chained Operation, signed by the sending DSA, which allows authentication of the sending DSA.

C.6 *Chaining through intermediate DSAs*

This procedure would be effected by DSA "c" in the model depicted in Figure C-1/X.518. DSA "c" will discard the signature provided by the sending DSA (DSA "b" in Figure C-1/X.518), and will modify the Chaining Argument, as described in 12.3 of this Recommendation. DSA "c" shall then combine the modified Chaining Argument with the signed Operation Argument, and sign the result to create a modified signed Chained Operation. This can be represented by:

c{ChAw', a{OA}} = the Chained Operation signed by DSA "c"

where

ChAw' = modified Chaining Argument.

This procedure would be effected by DSA "c" in the model depicted in Figure C-1/X.518. DSA "c" upon the nature of the operation, and upon the type of knowledge held, DSA "c" may perform request decomposition prior to chaining or multicasting any resultant operation(s). This has been represented in Figure C-1/X.518 by DSA "c" sending operations to DSA "d" and DSA "e"; in each case the authentication procedure is identical.

C.7 *Results authentication*

The results authentication service is requested by an initiator of a directory operation using the **signed** option within the **protectionRequest SecurityParameter**. In providing a response to such a request a DSA may optionally decide whether or not to sign any or all of the results; the results authentication service does not provide for the authentication of error responses.

Within the context of a particular DSA processing results from an arbitrary number of DSAs (each of which are associated with a particular service request) the following distinct cases are possible:

- the DSA provides a complete set of results for an operation without the need to perform any collating function (represented by DSA "d" and DSA "e" in Figure C-1/X.518);
- the DSA collates local results (sourced by this DSA) with the results from one or more other DSAs (represented by DSA "c" in Figure C-1/X.518);
- the DSA chains a result from a DSA to either another DSA or a DUA and does not contribute to the result set as it does so (represented by DSA "b" in Figure C-1/X.518).

C.7.1 *DSA results - no collation*

This paragraph addresses the role of a DSA in being the sole source of results to a particular operation request, i.e. the DSA has no collation function to perform. The paragraph considers the case for both the DSP and the DAP.

C.7.1.1 *DSP*

The DSA can choose to perform either of the following procedures:

- return the results unsigned, this can be represented by:

  ChR,OR = Chained Operation Result (unsigned)

  where

  ChR = Chaining Results

  OR = Operation Result;
- sign only the Operation Result, this can be represented by:

  ChR, d(OR) = Operation Result signed by DSA "d";
- sign only the Chained Operation Result, which can be represented as:

  d (ChR, OR) = Chained Operation Result signed by DSA "d"
- sign both the Operation Result and the Chained Operation Result, which can be represented by:

  d{ChR, D{OR}} = Operation Result and Chained Operation Result signed by DSA "d".

Note - For the case where the Operation Result is signed, the signed result will be carried back to the initiator; for the case where the Chained Operation Result has been signed, the receiving DSA will have to discard the signature in order to modify the Chaining Results argument prior to forwarding the Chained Operation Result.

C.7.1.2 *DAP*

This is fully described in Recommendation X.511, a summary is reproduced here for completeness.

The DSA can choose to either return the results unsigned, which can be represented by:

OR = Operation Result

or, signed, which can be represented by:

d{OR} = Operation Result signed by DSA "d".

C.7.2 *DSA results - collation included*

This paragraph addresses the role of a DSA in returning the result of particular service requests where collation and integration of results from other DSAs is a necessary prerequisite. The paragraph considers the case for both the DSP and the DAP.

C.7.2.1 *DSP*

Recognizing that zero or more results received from other DSAs may be signed, this procedure enables a DSA to collate and integrate the results and sign zero or more constituent parts of the composite result and optionally, sign the composite result as a whole.

C.7.2.1.1 *Production of the chaining results argument*

This procedure requires that a DSA (represented by DSA "c" in Figure C-1/X.518) remove all of the Chained Operation Result signatures from the results received from external DSAs (DSA "d" and DSA "e" in Figure C-1/X.518). DSA "c" then possesses a set of unsigned Chaining results, a set of signed Operation Results, and a set of unsigned Operation Results.

All the Chaining Results are manipulated as described in 12.4 of this Recommendation to create a single modified Chaining Result, denoted by:

i) ChRw' = modified Chaining Results.

C.7.2.1.2 *Unsigned locally derived result*

If the DSA does not wish to sign the locally generated results, the set of unsigned Operation Results are merged with the local result to form a modified set of Operation Results, denoted by:

ORw' = Merged Operation Result.

The complete set of Operation Results is then the union of the set of externally signed Operation Results denoted by:

d{OR}, e{OR} ...

and the Merged Operation Result, collectively denoted by:

(ii) ORw', d{OR}, e{OR} ...= Operation Result.

C.7.2.1.3 *Signed locally derived result*

If the DSA does wish to sign the locally generated results, then the externally generated set of unsigned Operation Results are first merged together. The complete set of Operation Results is then the union of the locally signed set of Operation Results denoted by C{OR}, the merged set of externally unsigned Operation Results denoted by, OR", and the set of externally signed Operation Results denoted by:

d{OR}, e{OR}, ..., which are collectively denoted as:

(iii) c{OR}, ORw", d{OR}, e{OR}, ... = Operation Result.

C.7.2.1.4 *Unsigned chained operation result*

If the DSA does not wish to sign the Chained Operation Result, then the latter will comprise the Chaining Results (identified in (i) above) added to the Operation Result identified in either (ii) or (iii) above, collectively, these are denoted by:

either:

ChRw', ORw', d{OR}, e{OR}, ... = Chained Operation Result (unsigned).

or,

ChRw', c{OR}, OR", d{OR}, e{OR}, ... = Chained Operation Result (unsigned) and Operation Result signed by DSA "c".

C.7.2.1.5 *Signed chained operation result*

If the DSA does wish to sign the Chained Operation Result, then the result will comprise the Chaining Results (identified in (i) above) added to the Operation Result (identified in either (ii) or (iii) above), collectively denoted as:

either:

c{ChRw', ORw', d{OR}, e{OR}, ...} = Chained Operation Result signed by DSA "c"

or,

c{ChRw', c{OR}, ORw", d{OR}, e{OR}, ...} = Chained Operation Result and Operation Result signed by DSA "c".

C.7.2.2 *DAP*

The procedure is very similar to that described in C.7.2.1, with the exception that the Chaining Results argument is not passed in the DAP.

C.7.3 *DSA chained results*

This paragraph addresses the procedures to be effected by a DSA in chaining an operation result back to the requestor, DSA or DUA, within the DSP and DAP respectively.

C.7.3.1 *DSP*

The DSA initially removes the signature (if one exists) from the Chained Operation Result. It then manipulates the Chaining Results argument as described in this Recommendation, to produce a modified Chaining Results argument. The latter is then merged back with the Operation Result argument to produce a modified Chained Operation Result. Finally, the DSA may optionally sign the Chained Operation Result before passing it to the next DSA in the chain.

C.7.3.2 *DAP*

A DSA (represented by DSA "b" in Figure C-1/X.518) first removes the signature (if one exists) from the Chained Operation Result. It then analyses and discards the Chaining Results argument and, finally, it optionally signs the remaining Operation Result argument before passing the result to the DUA.

ANNEX D

(to Recommendation X.518)

Distributed directory object identifiers

This Annex is part of the Recommendation.

This Annex includes all of the ASN.1 object identifiers contained in this Recommendation in the form of the ASN.1 module **DistributedDirectoryObjectIdentifiers**.

```
DistributedDirectoryObjectIdentifiers    {joint-iso-ccitt ds(5) modules(1)
                                          distributedDirectoryObjectIdentifiers(13)}

DEFINITION  ::=

BEGIN

EXPORTS
        id-ot-dsa, id-pt-chainedRead, id-pt-chainedSearch, id-pt-chainedModify;

IMPORTS
        id-ot, id-pt
            FROM    UsefulDefinitions {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)};

-- objects --

id-ot-dsa OBJECT IDENTIFIER  ::=  {id-ot 3}

-- part types --

id-pt-chainedRead       OBJECT IDENTIFIER       ::=     {id-pt 4}
id-pt-chainedSearch     OBJECT IDENTIFIER       ::=     {id-pt 5}
id-pt-chainedModify     OBJECT IDENTIFIER       ::=     {id-pt 6}

END
```