Groupe de travail Réseau
Request for Comments : 4511
RFC rendues obsolètes : 2251, 2830, 3771
Catégorie : Normes
juin 2006

J. Sermersheim, éditeur, Novell, Inc.

Traduction Claude Brière de L'Isle
décembre 2006

Protocole léger d'accès à un répertoire (LDAP) : le protocole

Statut de ce mémo

Le présent document spécifie un protocole de normalisation Internet pour la communauté de l'Internet, qui appelle à la discussion et à des suggestions pour son amélioration. Prière de se reporter à l'édition en cours des "Normes de protocole officielles de l'Internet" (STD 1) sur l'état de la normalisation et le statut de ce protocole. La distribution du présent mémo n'est soumise à aucune restriction.

Notice de Copyright

Résumé

Le présent document décrit les éléments de protocole, ainsi que leur sémantique et leurs codages, du Protocole léger d'accès à un répertoire, (LDAP, *Lightweight Directory Access Protocol*). LDAP fournit l'accès à des services distribués de répertoires qui agissent conformément aux modèles de données et services de X.500. Ces éléments de protocole se fondent sur ceux décrits dans le protocole d'accès d'annuaire (DAP) X.500.

Table des matières

1 Introduction

Le répertoire est "une collection de systèmes ouverts coopérant pour fournir des services d'annuaire" [X.500]. Un utilisateur de répertoire, qui peut être une personne ou une autre entité, accède au répertoire à travers un client (ou agent utilisateur de répertoire (DUA, *Directory User Agent*). Le client, au nom de l'utilisateur de répertoire, interagit avec un ou plusieurs serveurs (ou agents systèmes de répertoire (DSA, *Directory System Agents*)). Les clients interagissent avec les serveurs en utilisant un protocole d'accès de répertoire.

Le présent document précise les éléments de protocole du protocole léger d'accès à des répertoires (LDAP), ainsi que leur sémantique. Après la description of éléments de protocole, il décrit la façon dont les éléments de protocole sont codés et transférés.

1.1 Relations avec les autres spécifications LDAP

Le présent document faits partie intégrante de la spécification technique LDAP [RFC4510], qui rend obsolète la spécification technique LDAP précédemment définie, RFC 3377, dans sa totalité.

Le présent document, conjointement avec les [RFC4510], [RFC4513], et [RFC4512], rend obsolète la RFC 2251 dans sa totalité. Le paragraphe 3.3 est rendu obsolète par la [RFC4510]. Les paragraphes 4.2.1 (portions) et 4.2.2 sont rendues obsolètes par la [RFC4513]. Les paragraphes 3.2, 3.4, 4.1.3 (dernier alinéa), 4.1.4, 4.1.5, 4.1.5.1, 4.1.9 (dernier alinéa), 5.1, 6.1, et 6.2 (dernier alinéa) sont rendues obsolètes par la [RFC4512]. Le reste de la RFC 2251 est rendue obsolète par le présent document. L'Appendice C.1 résume les changements de substance dans le reste.

Le présent document rend obsolète la RFC 2830, Sections 2 et 4. Le reste de la RFC 2830 est rendu obsolète par la [RFC4513]. L'Appendice C.2 résume les changements de substance dans les sections restantes.

Le présent document rend aussi obsolète la RFC 3771 dans sa totalité.

2 Conventions

Les mots clé "DOIT", "NE DOIT PAS", "DEVRAIT", "NE DEVRAIT PAS", "PEUT", et "FACULTATIF" dans le présent document sont à interpréter comme décrit dans le BCP 14 [RFC2119].

Les noms de caractères dans le présent document utilisent la notation pour les codets et noms de la norme Unicode [Unicode]. Par exemple, la lettre "a" peut être représentée comme <U+0061> ou comme <LETTRE LATINE MINUSCULE A>.

Note : un glossaire des termes utilisés en Unicode se trouve dans [Glossary]. Des informations sur le modèle de codage de caractères Unicode se trouve dans [CharModel].

Le terme "connexion de transport" se réfère aux services de transport sous-jacents utilisés pour porter l'échange de protocole, ainsi que les associations établies par ces services.

Le terme "couche TLS" se réfère aux services de sécurité de la couche transport (TLS, *Transport Layer Security*) utilisés pour fournir des services de sécurité, ainsi que les associations établies par ces services.

Le terme "couche SASL" se réfère aux services de couche simple d'authentification et de sécurité (SASL, *Simple Authentication and Security Layer*) utilisés pour fournir des services de sécurité, ainsi que les associations établies par ces services.

Le terme "couche de message LDAP" se réfère aux services d'unité de protocole de données (PDU, *Protocol Data Unit*) de message LDAP utilisés pour fournir des services de répertoire, ainsi que les associations établies par ces services.

Le terme "session LDAP" se réfère à des services combinés (connexion de transport, couche TLS, couche SASL, couche de message LDAP) et leurs associations.

Voir au tableau de la Section 5 une illustration de ces quatre termes.

3 Modèle de protocole

Le modèle général adopté par le présent protocole est celui de clients effectuant des opérations de protocole sur des serveurs. Dans ce modèle, un client transmet une demande de protocole qui décrit les opérations à effectuer sur un serveur. Le serveur est alors responsable de la réalisation de la ou des opérations nécessaires dans le répertoire. A l'achèvement d'une opération, le serveur retourne normalement une réponse contenant les données appropriées au client demandeur.

Les opérations de protocole sont généralement indépendantes les unes des autres. Chaque opération est traitée comme une action atomique, laissant le répertoire dans un état cohérent.

Bien que les serveurs soient obligés de retourner des réponses à chaque fois que de telles réponses sont définies dans le protocole, il n'y a pas d'obligation pour un comportement synchronisé de la part ni des clients ni des serveurs. Les demandes et les réponses pour des opérations multiples peuvent généralement être échangées entre un client et un serveur dans n'importe quel ordre. Si nécessaire, un comportement synchronisé peut être commandé par les applications du client.

Le coeur des opérations de protocole définies dans le présent document peut être transposé à un sous ensemble du service abstrait d'annuaire X.500 (1993) [X.511]. Cependant, il n'y a pas de transposition bijective entre les opérations LDAP et les opérations du protocole d'accès de répertoire (DAP) X.500. Les mises en œuvre de serveur agissant en tant qu passerelles vers des répertoires X.500 peuvent avoir besoin de faire plusieurs demandes DAP pour servir une seule demande LDAP.

3.1 Relations entre opération et couche de message LDAP

Les opérations de protocole sont échangées à la couche de message LDAP. Lorsque la connexion de transport est fermée, toute opération non achevée à la couche de message LDAP est abandonnée (lorsque c'est possible) ou est achevée sans transmission de la réponse (lorsque leur abandon n'est pas possible). Aussi, lorsque la connexion de transport est fermée, le client NE DOIT PAS supposer qu'une opération de mise à jour non achevée a réussi ou échoué.

4 Éléments de protocole

Le protocole est décrit à l'aide de la notation de syntaxe abstraite n° 1 ([ASN.1], *Abstract Syntax Notation One*) et il est transféré en utilisant un sous-ensemble d'ASN.1, les règles de codage de base ([BER], *Basic Encoding Rules*). La Section 5 spécifie la façon dont les éléments de protocole sont codés et transférés.

Pour prendre en charge les extensions futures du protocole, l'extensibilité est implicite lorsqu'elle est permise selon ASN.1 (c'est-à-dire que les types, sequence, set, choice, et enumerated sont extensibles). De plus, les ellipses (...) ont été fournies dans les types ASN.1 qui sont explicitement extensibles comme exposé dans la [RFC4520]. A cause de extensibilité implicite, les clients et serveurs DOIVENT (sauf spécification contraire) ignorer les composants SEQUENCE en queue dont ils ne reconnaissent pas les étiquettes.

Les changements au protocole autres que par les mécanismes d'extension décrits ici exigent un numéro de version différent. Un client indique la version qu'il utilise au titre de la BindRequest, décrite au paragraphe 4.2. Si un client n'a pas envoyé un Bind, le serveur DOIT supposer que le client utilise la version 3 ou plus récente.

Les clients peuvent essayer de déterminer les versions de protocole qu'un serveur prend en charge en lisant l'attribut 'supportedLDAPVersion' d'après la DSE (DSA-Specific Entry) racine [RFC4512].

4.1 Éléments communs

La présente section décrit le format d'unité de données de protocole (PDU) d'enveloppe de message LDAP, LDAPMessage, ainsi que les définitions de type de données, qui sont utilisées dans des opérations de protocole.

4.1.1 Enveloppe de message

Pour les besoins des échanges de protocole, toutes les opérations de protocole sont encapsulées dans une enveloppe commune, le LDAPMessage, qui se définit comme suit :

```
LDAPMessage ::= SEQUENCE {
    messageID       MessageID,
    protocolOp      CHOICE {
        bindRequest           BindRequest,
        bindResponse          BindResponse,
        unbindRequest         UnbindRequest,
        searchRequest         SearchRequest,
        searchResEntry        SearchResultEntry,
        searchResDone         SearchResultDone,
        searchResRef          SearchResultReference,
        modifyRequest         ModifyRequest,
        modifyResponse        ModifyResponse,
        addRequest            AddRequest,
        addResponse           AddResponse,
        delRequest            DelRequest,
        delResponse           DelResponse,
        modDNRequest          ModifyDNRequest,
        modDNResponse         ModifyDNResponse,
        compareRequest        CompareRequest,
        compareResponse       CompareResponse,
        abandonRequest        AbandonRequest,
        extendedReq           ExtendedRequest,
        extendedResp          ExtendedResponse,
        ...,
        intermediateResponse  IntermediateResponse },
    controls       [0] Controls OPTIONAL }

MessageID ::= INTEGER (0 ..  maxInt)

maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --
```

Le type ASN.1 Controls est défini au paragraphe 4.1.11.

La fonction de LDAPMessage est de fournir une enveloppe contenant des champs communs nécessaires dans tous les échanges de protocole. Actuellement, les seuls champs communs sont le messageID et le controls.

Si le serveur reçoit un LDAPMessage de la part du client, dans lequel l'étiquette SEQUENCE de LDAPMessage ne peut pas être reconnue, le messageID ne peut pas être analysée, l'étiquette du protocolOp n'est pas reconnue comme une demande, ou les structures de codages ou les longueurs des champs de données se trouvent incorrects, le serveur DEVRAIT retourner l'avis de déconnexion décrit au paragraphe 4.4.1, avec le code de résultat *(resultCode)* réglé à protocolError, et DOIT immédiatement terminer la session LDAP comme décrit au paragraphe 5.3.

Dans les autres cas où le client ou le serveur ne peuvent pas analyser un PDU LDAP, ils DEVRAIENT terminer aussitôt la session LDAP (paragraphe 5.3) car des communications ultérieures (y compris de fournir un avis) pourrait

se révéler pernicieux. Autrement, les mises en œuvre de serveur DOIVENT retourner une réponse appropriée à la demande, avec le code de résultat établi à protocolError.

4.1.1.1 MessageID

Toutes les enveloppes de LDAPMessage encapsulant des réponses contiennent la valeur de messageID du LDAPMessage de la demande correspondante.

Le messageID d'une demande DOIT avoir une valeur autre que zéro différente du messageID de toute autre demande en cours dans la même session LDAP. La valeur zéro est réservée pour le message de notification non sollicité.

Les clients normaux incrémentent un compteur pour chaque demande.

Un client NE DOIT PAS envoyer une demande avec le même messageID qu'une demande précédente dans la même session LDAP sauf s'il peut être déterminé que le serveur n'est plus en train de servir la demande précédente (par exemple, après la réception de la réponse finale, ou l'achèvement d'un Bind suivant). Autrement, le comportement est indéfini. Dans ce but, noter que Abandon et les opérations abandonnées avec succès n'envoient pas de réponse.

4.1.2 Types de chaînes

La chaîne LDAPString est une convention de notation pour indiquer que, bien que les chaînes de type LDAPString se codent comme des types CHAINE D'OCTET ASN.1, on utilise l'ensemble de caractères [ISO10646] (un sur ensemble de [Unicode]), codé selon l'algorithme UTF-8 de la [RFC3629]. Noter que les caractères Unicode U+0000 à U+007F sont les mêmes que les caractères ASCII 0 à 127, respectivement, et ont le même codage UTF-8 d'un seul octet. Les autres caractères Unicode ont un codage UTF-8 à plusieurs octets.

LDAPString ::= CHAINE D'OCTET – codé en UTF-8, caractères [ISO10646]

L'identifiant LDAPOID est une convention de notation pour indiquer que la valeur permise de cette chaîne est une représentation en décimal à séparation des octets par des points (codé en UTF-8) d'un IDENTIFIANT D'OBJET. Bien qu'un identifiant LDAPOID soit codé comme une CHAINE D'OCTET, les valeurs sont limitées à la définition du <numericoid> donné au paragraphe 1.4 de la [RFC4512].

LDAPOID ::= OCTET STRING -- Constrained to <numericoid> [RFC4512]

Par exemple,

1.3.6.1.4.1.1466.1.2.3

4.1.3 Nom distinctif et nom distinctif relatif

Un nom distinctif LDAPDN se définit comme étant la représentation d'un nom distinctif (DN) après codage, conformément à la spécification de la [RFC4514].

LDAPDN ::= LDAPString -- Forcé à <distinguishedName> [RFC4514]

Un nom distinctif RelativeLDAPDN se définit comme étant la représentation d'un nom distinctif relatif (RDN, *Relative Distinguished Name*) après codage conformément à la spécification de la [RFC4514].

RelativeLDAPDN ::= LDAPString -- Forcé à <name-component> [RFC4514]

4.1.4 Descriptions d'attribut

La définition et les règles de codage pour les descriptions d'attribut sont définies au paragraphe 2.5 de la [RFC4512]. En bref, une description d'attribut est un type d'attribut et zéro, une ou plusieurs options.

AttributeDescription ::= LDAPString -- Forcé à <attributedescription> [RFC4512]

4.1.5 Valeur d'attribut

Un champ de type AttributeValue est une CHAINE D'OCTET contenant une valeur d'attribut codée. La valeur d'attribut est codée conformément à la définition de codage spécifique de LDAP de sa syntaxe correspondante. Les définitions de codage spécifiques de LDAP pour les différents types de syntaxe et d'attribut pourront être trouvés dans d'autres documents et en particulier dans la [RFC4517].

```
AttributeValue ::= CHAINE D'OCTET
```

Noter qu'il n'y a pas de limite définie pour la taille de ce codage ; et donc, les valeurs du protocole peuvent inclure des valeurs d'attribut de plusieurs méga octets (par exemple, des photographies).

Des valeurs d'attribut peuvent être définies avec une syntaxe arbitraire et non imprimable. Les mises en œuvre NE DOIVENT PAS afficher ou essayer de décoder une valeur d'attribut si sa syntaxe n'est pas connue. La mise en œuvre peut essayer de découvrir le sous-schéma de l'entrée de source et de restituer les descriptions de 'attributeTypes' à partir de là [RFC4512].

Les clients DOIVENT seulement envoyer des valeurs d'attribut dans une demande valide conformément à la syntaxe définie pour les attributs.

4.1.6 Assertion de valeur d'attribut

La définition du type AttributeValueAssertion (AVA) est semblable à celle de la norme d'annuaire X.500. Elle contient une description d'attribut et une valeur d'assertion de règle de correspondance ([RFC4512] paragraphe 4.1.3) convenable pour ce type. Les éléments de ce type sont normalement utilisés pour affirmer que la valeur dans assertionValue correspond à une valeur d'un attribut.

```
AttributeValueAssertion ::= SEQUENCE {
        attributeDesc   AttributeDescription,
        assertionValue  AssertionValue }
AssertionValue ::= OCTET STRING
```

La syntaxe de AssertionValue dépend du contexte de l'opération LDAP effectuée. Par exemple, la syntaxe la règle de correspondance EQUALITY pour un attribut est utilisée lorsqu'on effectue une opération Compare. Souvent c'est la même syntaxe qu'utilisée pour les valeurs du type d'attribut, mais dans certains cas, la syntaxe d'assertion diffère de la syntaxe de valeur. Voir par exemple objectIdentiferFirstComponentMatch dans la [RFC4517].

4.1.7 Attribute et PartialAttribute

Les attributs et les attributs partiels consistent en une description d'attribut et en valeurs d'attribut. Un PartialAttribute permet des valeurs zéro, alors que Attribute exige au moins une valeur.

```
PartialAttribute ::= SEQUENCE {
    type       AttributeDescription,
    vals       SET OF value AttributeValue }
Attribute ::= PartialAttribute(WITH COMPONENTS {
    ...,
    vals (SIZE(1..MAX))})
```

Deux valeurs de l'attribut ne doivent pas être équivalentes comme décrit au paragraphe 2.2 de la [RFC4512]. L'ensemble des valeurs d'attribut n'est pas ordonné. Les mises en œuvre NE DOIVENT PAS supposer que l'ordre soit répétable.

4.1.8 Identifiant de règle de correspondance

Les règles de correspondance sont définies au paragraphe 4.1.3 de la [RFC4512]. Une règle de corerspondance est identifiée dans le protocole par la représentation imprimable de son <numericoid> ou par un de ses descripteurs abrégés [RFC4512], par exemple, 'caseIgnoreMatch' ou '2.5.13.2'.

```
MatchingRuleId ::= LDAPString
```

4.1.9 Message de résultat

Le résultat LDAPResult est la construction utilisée dans ce protocole pour retourner les indications de succès ou d'échec de la part des serveurs aux clients. Pour diverses demandes, les serveurs vont retourner des réponses contenant les éléments trouvés dans LDAPResult pour indiquer l'état final de la demande d'opération de protocole.

```
LDAPResult ::= SEQUENCE {
        resultCode      ENUMERATED {
        success                         (0),
        operationsErro                  (1),
        protocolError                   (2),
        timeLimitExceeded               (3),
        sizeLimitExceeded               (4),
        compareFalse                    (5),
        compareTrue                     (6),
        authMethodNotSupported          (7),
        strongerAuthRequired            (8),
                                        -- 9 réservé --
        referral                        (10),
        adminLimitExceeded              (11),
        unavailableCriticalExtension    (12),
        confidentialityRequired         (13),
        saslBindInProgress              (14),
        noSuchAttribute                 (16),
        undefinedAttributeType          (17),
        inappropriateMatching           (18),
        constraintViolation             (19),
        attributeOrValueExists          (20),
        invalidAttributeSyntax          (21),
                                        -- 22-31 inutilisé --
        noSuchObject                    (32),
        aliasProblem                    (33),
        invalidDNSyntax                 (34),
                                        -- 35 réservé pour undefined isLeaf --
        aliasDereferencingProblem       (36),
                                        -- 37-47 inutilisé --
        inappropriateAuthentication     (48),
        invalidCredentials              (49),
        insufficientAccessRights        (50),
        busy                            (51),
        unavailable                     (52),
        unwillingToPerform              (53),
        loopDetect                      (54),
                                        -- 55-63 inutilisé --
        namingViolation                 (64),
        objectClassViolation            (65),
        notAllowedOnNonLeaf             (66),
        notAllowedOnRDN                 (67),
        entryAlreadyExists              (68),
        objectClassModsProhibited       (69),
                                        -- 70 réservé pour CLDAP --
        affectsMultipleDSAs             (71),
                                        -- 72-79 inutilisé --
        other                           (80),
... },
matchedDN                               LDAPDN,
diagnosticMessage                       LDAPString,
```

referral [3] Referral OPTIONAL }

L'énumération des codes resultCode est extensible comme défini au paragraphe 3.8 de la [RFC4520]. La signification des codes de résultat énumérés est donnée à l'Appendice A. Si un serveur détecte plusieurs erreurs pour une opération, un seul code de résultat est retourné. Le serveur devrait retourner le code de résultat qui indique le mieux la nature de l'erreur rencontrée. Les serveurs peuvent retourner des codes de résultat substitués pour empêcher une divulgation non autorisée.

Le champ diagnosticMessage de cette construction peut, au choix du serveur, être utilisé pour retourner une chaîne contenant un message de diagnostic textuel, lisible par l'homme (les caractères de commande terminaux et de formatage de page devraient être évités). Comme ce message de diagnostic n'est pas normalisé, les mises en œuvre NE DOIVENT PAS se fier aux valeurs retournées. Les messages de diagnostic ajoutent normalement des informations supplémentaires au resultCode. Si le serveur choisit de ne pas retourner un diagnostic textuel, le champ diagnosticMessage DOIT être vide.

Pour certains codes de résultat (ce sont normalement, mais pas exclusivement noSuchObject, aliasProblem, invalidDNSyntax, et aliasDereferencingProblem), le champ matchedDN est réglé (sous réserve des contrôles d'accès) au nom de la dernière entrée (objet ou alias) utilisé en trouvant l'objet cible (ou de base). Ce sera une forme tronquée du nom fourni, ou, si un alias a été déréférencé en tentant de localiser l'entrée, du nom résultant. Autrement, le champ matchedDN est vide.

4.1.10 Renvoi de référence

Le code de résultat renvoi de référence (*referral*) indique que le serveur contacté ne peut pas ou ne veut pas effectuer l'opération et qu'un ou plusieurs autres serveurs peuvent en être capables. Les raisons pour cela peuvent être que :

- l'entrée cible de la demande n'est pas tenue localement, mais le serveur a connaissance de son existence possible ailleurs,
- l'opération est soumise à des restrictions sur ce serveur – peut-être dues à une copie en lecture seule de l'entrée à modifier

Le champ referral est présent dans un LDAPResult si le resultCode est réglé à referral, et est absent de tous les autres codes de résultat. Il contient une ou plusieurs références à un ou plusieurs serveurs ou services auxquels on peut accéder via LDAP ou d'autres protocoles. Des renvois de référence peuvent être retournés en réponse à toute demande d'opération (exceptéUnbind et Abandon, qui n'ont pas de réponse). Au moins un URI DOIT être présent dans le Referral.

Durant une opération Search, après localisation de l'objet baseObject, et l'évaluation des entrées, le renvoi de référence n'est pas retourné. Au lieu de cela, les références de continuation, décrites au paragraphe 4.5.3, sont retournées lorsque d'autres serveurs doivent être contactés pour achever l'opération.

Referral ::= SEQUENCE SIZE (1..MAX) OF uri URI

URI ::= LDAPString -- limité aux caractères permis dans les URI

Si le client souhaite continuer l'opération, il contacte un des services pris en charge trouvés dans le renvoi de références. Si plusieurs URI sont présents, le client suppose que tout URI pris en charge peut être utilisé pour continuer l'opération.

Les clients qui suivent les renvois de références DOIVENT s'assurer qu'ils ne font pas une boucle entre les serveurs. Ils NE DOIVENT PAS contacter de façon répétée le même serveur pour la même demande avec les mêmes paramètres. Certains clients utilisent un compteur qui est incrémenté chaque fois que survient un traitement de renvoi de référence pour une opération, et cette sorte de clients DOIT être capable de traiter au moins dix renvois de référence enchâssés lors du traitement de l'opération.

Un URI pour un serveur qui met en œuvre LDAP et est accessible via TCP/IP (v4 ou v6) [RFC793][RFC791] est écrit comme un URL LDAP conformément à la [RFC4516].

Les valeurs de Referral qui sont des URL LDAP suivent les règles ci-après :

- Si un alias a été déréférencé, la partie <dn> de l'URL LDAP DOIT être présente, avec le nouveau nom d'objet cible.
- Il est RECOMMANDÉ que la partie <dn> soit présente pour éviter les ambiguïtés.
- Si la partie <dn> est présente, le client utilise ce nom dans sa demande suivante pour continuer l'opération, et si elle n'est pas présente, le client utilise le même nom que dans la demande d'origine.
- Certains serveurs (par exemple, qui participent à une indexation distribuée) peuvent fournir dans un URL un filtre différent d'un renvoi de références pour une opération Search.
- Si la partie <filtre> de l'URL LDAP est présente, le client utilise ce filtre dans sa demande suivante pour continuer cette recherche *(Search)*, et si elle n'est pas présente, le client utilise le même filtre que celui utilisé pour Search.
- Pour Search, il est RECOMMANDÉ que la partie <scope> soit présente pour éviter les ambiguïtés.
- Si la partie <scope> manque, le scope du Search d'origine est utilisé par le client pour continuer l'opération.
- Les autres aspects de la nouvelle demande peuvent être les mêmes que ceux de la demande qui a généré le renvoi de références, ou différents.

D'autres sortes d'URI peuvent être retournés. La syntaxe et la sémantique de tels URI fra l'objet de spécifications futures. Les clients peuvent ignorer les URI qu'ils ne prennent pas en charge.

Les caractères codés en UTF-8 qui apparaissent dans la représentation de chaîne d'un nom distinctif, filtre de recherche, ou autres champs de la valeur de renvoi de références peuvent n'être pas légaux pour les URI (par exemple, les espaces) et DOIVENT être codés en pourcentage en utilisant la méthode % de la [RFC3986].

4.1.11 Controls

Controls fournit un mécanisme par lequel la sémantique et les arguments des opérations LDAP existantes peuvent être étendus. Un ou plusieurs controls peuvent être attachés à un seul message LDAP. Un control n'affecte que la sémantique du message auquel il est attaché.

Les controls envoyés par les clients sont appelés 'contrôles de demande', et ceux envoyés par les serveurs sont appelés 'contrôles de réponse'.

```
Controls ::= SEQUENCE OF control Control

Control ::= SEQUENCE {
    controlType             LDAPOID,
    criticality             BOOLEAN DEFAULT FALSE,
    controlValue            OCTET STRING OPTIONAL }
```

Le champ controlType est la représentation en décimal à séparation des octets par des points d'un IDENTIFIANT D'OBJET qui identifie le contrôle de façon univoque. Cela donne une dénomination non ambiguë des contrôles. Souvent, le ou les contrôles de réponse sollicités par un contrôle de demande partagent les valeurs de controlType avec le contrôle de la demande.

Le champ criticality n'a de signification que dans les contrôles attachés à des messages de demande (sauf UnbindRequest). Pour les contrôles attachés à des messages de réponse et pour UnbindRequest, le champ criticality DEVRAIT être FALSE, et DOIT être ignoré par l'homologue de protocole receveur. Une valeur de TRUE indique qu'il lui est inacceptable d'effectuer l'opération sans appliquer la sémantique de contrôle. Précisément, le champ criticality s'applique comme suit :

- Si le serveur ne reconnaît pas le type de contrôle, il détermine s'il n'est pas approprié pour l'opération, ou si il n'est pas autrement réticent à effectuer l'opération avec le contrôle, et si le champ criticality est TRUE, le serveur NE DOIT PAS effectuer l'opération, et pour les opérations qui ont un message de réponse, il DOIT retourner le code de résultat réglé à unavailableCriticalExtension.
- Si le serveur ne reconnaît pas le type de contrôle, il détermine si il n'est pas approprié pour l'opération, ou si il n'est pas autrement réticent à effectuer l'opération avec le contrôle, et si le champ criticalité est FALSE, le serveur DOIT ignorer le contrôle.
- Indépendamment de criticality, si un contrôle s'applique à une opération, il est appliqué de façon cohérente et impartiale à l'opération toute entière.

La valeur controlValue peut contenir des informations associées au controlType. Son format est défini par la spécification du contrôle. Les mises en œuvre DOIVENT être prêtes à traiter des contenus arbitraires de la chaîne

d'octets controlValue, y compris des octets zéro. Elle n'est absente que si aucune information de valeur n'est associée à un control de son type. Lorsqu'une controlValue est définie en termes d'ASN.1, et codée en BER conformément au paragraphe 5.1, elle suit aussi les règles d'extensibilité de la Section 4.

Les serveurs font la liste des controlType des contrôles de demande qu'ils reconnaissent dans l'attribut 'supportedControl' dans le DSE racine (paragraphe 5.1 de la [RFC4512]).

Les contrôles NE DEVRAIENT PAS être combinés si la sémantique de combinaison n'a pas été spécifiée. La sémantique des combinaisons de control, si elle est spécifiée, se trouve généralement dans la spécification de control publiée le plus récemment. Lorsqu'on rencontre une combinaison de controls dont la sémantique est invalide, non spécifiée (ou inconnue), le message est considéré comme mal formé ; et donc, l'opération échoue avec protocolError. Les contrôles avec une criticality de FALSE peuvent être ignorés afin d'arriver à une combinaison valide. De plus, sauf si une sémantique dépendante de l'ordre est donnée dans une spécification, l'ordre d'une combinaison de controls dans la SEQUENCE est ignoré. Lorsque l'ordre est à ignorer mais ne peut pas être ignoré par le serveur, le message est considéré comme mal formé, et l'opération échoue avec protocolError. Là encore, les contrôles avec une criticality de FALSE peuvent être ignorés afin d'arriver à une combinaison valide.

Le présent document ne spécifie aucun contrôle. Les contrôles peuvent être spécifiés dans d'autres documents. Les documents qui précisent les extensions de contrôle sont à fournir pour chaque contrôle :

- l'IDENTIFIANT D'OBJET alloué au contrôle,
- l'indication de la valeur que l'envoyeur devrait fournir pour le champ criticality (note : la sémantique du champ criticality qui est définie ci-dessus ne devrait pas être altérée par la spécification du contrôle),
- si le champ controlValue est présent, et si oui, le format de son contenu,
- la sémantique du contrôle, et
- facultativement, la sémantique concernant la combinaison du contrôle avec d'autres contrôles.

4.2 Opération Bind

La fonction de l'opération Bind est de permettre d'échanger les informations d'authentification entre le client et le serveur. L'opération Bind devrait être pensée comme l'opération "authenticate". La sémantique de fonctionnement, d'authentification, et de ce qui se rapporte à la sécurité de cette opération est donnée dans la [RFC4513].

La demande Bind se définit comme suit :

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
    version             INTEGER (1 ..  127),
    name                LDAPDN,
    authentication        AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
    simple              [0] OCTET STRING,
                        -- 1 et 2 reserved
    sasl                [3] SaslCredentials,
    ...  }

SaslCredentials ::= SEQUENCE {
    mechanism             LDAPString,
    credentials           OCTET STRING OPTIONAL }
```

Les champs de la demande BindRequest sont :

- version : Un numéro de version indique la version du protocole à utiliser à la couche de message LDAP. Le présent document décrit la version 3 du protocole. Il n'y a pas de négociation de version. Le client règle ce champ à la version qu'il désire. Si le serveur ne prend pas en charge la version spécifiée, il DOIT répondre par un BindResponse où le code de résultat est réglé à protocolError.

- name : S'il n'est pas vide, c'est le nom de l'objet de répertoire auquel le client souhaite se lier. Ce champ peut prendre une valeur nulle (un chaîne de longueur zéro) pour les besoins des liens anonymes ([RFC4513], paragraphe

5.1) ou lors de l'utilisation de l'authentification SASL [RFC4422] ([RFC4513] paragraphe 5.2). Lorsque le serveur essaye de localiser l'objet nommé, il NE DOIT PAS effectuer de déréférencement d'alias.

- authentication : Ce sont les informations utilisées dans l'authentification. Ce type est extensible comme défini au paragraphe 3.7 de la [RFC4520]. Les serveurs qui ne prennent pas en charge le choix fourni par un client retournent une BindResponse avec le code de résultat réglé à authMethodNotSupported.

Les mots de passe textuels (consistant en une séquence de caractères avec un ensemble de caractères et un codage connus) transférés au serveur en utilisant le AuthenticationChoice simple DOIVENT être transférés codés en UTF-8 [RFC3629] [Unicode]. Avant le transfert, les clients DEVRAIENT préparer des mots de passe textuels comme des chaînes "query" en appliquant le profil SASLprep [RFC4013] à l'algorithme stringprep [RFC3454]. Les mots de passe consistant en autres données (comme des octets aléatoires) NE DOIVENT PAS être altérés. La détermination du caractère textuel ou non d'un mot de passe est l'affaire du client local.

4.2.1 Traitement de la demande Bind

Avant de traiter une BindRequest, toutes les opérations non achevées DOIVENT soit être complétées soit abandonnées. Le serveur peut attendre que les opérations non achevées se terminent, ou les abandonner. Le serveur procède ensuite à l'authentification du client par un processus en une seule étape ou en plusieurs étapes. Chaque étape exige que le serveur retourne une BindResponse pour indiquer l'état de l'authentification.

Après l'envoi d'une BindRequest, les clients NE DOIVENT PAS envoyer d'autres PDU LDAP jusqu'à la réception de la BindResponse. De même, les serveurs NE DEVRAIENT PAS traiter ou répondre aux demandes reçues pendant le traitement d'une BindRequest.

Si le client ne s'est pas lié avant d'envoyer une demande et reçoit une operationsError à cette demande, il peut alors envoyer une BindRequest.Si celle-ci échoue aussi ou si le client choisit de ne pas se lier à la session LDAP existante, il peut terminer la session LDAP, la rétablir, et recommencer en envoyant d'abord une BindRequest. Cela aidera à interopérer avec les serveurs qui mettent en œuvre d'autres versions de LDAP.

Les clients peuvent envoyer plusieurs demandes Bind pour changer les associations d'authentification et/ou de sécurité ou terminer un processus Bind multi-étapes. L'authentification provenant de liens antérieurs sera ignorée.

Pour certains mécanismes d'authentification SASL, il peut être nécessaire que le client invoque plusieurs fois la BindRequest ([RFC4513], paragraphe 5.2). Les clients NE DOIVENT PAS invoquer d'opérations entre deux demandes Bind faites au titre d'un Bind multi-étapes.

Un client peut interrompre une négociation SASL en envoyant une BindRequest avec une valeur différente dans le champ mécanisme de SaslCredentials, ou un AuthenticationChoice autre que sasl.

Si le client envoie une BindRequest avec le champ mécanisme sasl comme une chaîne vide, le serveur DOIT retourner une BindResponse avec le code de résultat réglé à authMethodNotSupported. Cela permettra au client d'interrompre une négociation si il souhaite essayer à nouveau avec le même mécanisme SASL.

4.2.2 Réponse Bind

La réponse Bind se définit comme suit :

```
BindResponse ::= [APPLICATION 1] SEQUENCE {
    COMPONENTS OF LDAPResult,
    serverSaslCreds    [7] OCTET STRING OPTIONAL }
```

La BindResponse consiste simplement en une indication de la part du serveur de l'état de la demande d'authentification du client.

Une opération Bind réussie est indiquée par une BindResponse avec un resultCode réglé à succès. Autrement, un code de résultat approprié est établi dans la BindResponse. Pour BindResponse, le code de résultat protocolError peut être utilisé pour indiquer que le numéro de version fourni par le client n'est pas pris en charge.

Si le client reçoit une BindResponse où le resultCode est réglé à protocolError, il doit supposer que le serveur ne prend pas en charge cette version de LDAP. Alors que le client peut être capable de fonctionner avec une autre version de ce protocole (qui peut ou non exiger la clôture et le rétablissement de la connexion de transport), la façon de poursuivre avec cette autre version de ce protocole sort du domaine d'application du présent document. Les clients qui ne peuvent ou ne veulent pas poursuivre DEVRAIENT terminer la session LDAP.

Le champ serverSaslCreds est utilisé au titre d'un mécanisme de liaison défini par SASL pour permettre au client de s'authentifier auprès du serveur avec lequel il communique, ou d'effectuer une authentification "mise en cause-réponse". Si le client se lie avec le choix simple, ou si le mécanisme SASL n'exige pas que le serveur retourne l'information au client, ce champ NE DOIT PAS alors être inclus dans la BindResponse.

4.3 Opération Unbind

La fonction de l'opération Unbind est de terminer une session LDAP. L'opération Unbind n'est pas l'antithèse de l'opération Bind comme son nom semble l'impliquer. La dénomination de ces opérations a un caractère historique. L'opération Unbind devrait être vue comme l'opération "quitter".

L'opération Unbind se définit comme suit :

```
UnbindRequest ::= [APPLICATION 2] NULL
```

Le client, à la transmission de UnbindRequest, et le serveur, à réception de UnbindRequest, doivent terminer de bon gré la session LDAP comme décrit au paragraphe 5.3. Les opérations non achevées sont traitées comme spécifié au paragraphe 3.1.

4.4 Notification non sollicitée

Une notification non sollicitée est un LDAPMessage envoyé du serveur au client, qui n'est pas une réponse à un LDAPMessage reçu par le serveur. Elle est utilisée pour signaler une condition extraordinaire dans le serveur ou dans la session LDAP entre le client et le serveur. La notification est de nature informative, et le serveur n'attendra aucune réponse en retour de la part du client.

La notification non sollicitée est structurée comme un LDAPMessage dans lequel le messageID est zéro et le protocolOp est réglé au choix extendedResp en utilisant le type ExtendedResponse (voir au paragraphe 4.12). Le champ responseName de ExtendedResponse contient toujours un identifiant LDAPOID unique pour cette notification.

Une notification non sollicitée (Avis de déconnexion) est définie dans le présent document. La spécification d'une notification non sollicitée consiste en :
- un IDENTIFIANT D'OBJET alloué à la notification (à spécifier dans le responseName),
- le format du contenu de la responseValue (s'il en est),
- les circonstances qui causeront l'envoi de la notification, et
- la sémantique du message.

4.4.1 Avis de déconnexion

Cette notification peut être utilisée par le serveur pour aviser le client que le serveur est sur le point de terminer la session LDAP de sa propre initiative. Cette notification est destinée à aider les clients à distinguer entre une condition exceptionnelle de serveur et un échec réseau transitoire. Noter que cette notification n'est pas une réponse à un Unbind demandé par le client. Les opérations non achevées sont traitées comme spécifié au paragraphe 3.1.

Le responseName est 1.3.6.1.4.1.1466.20036, le champ responseValue est absent, et le resultCode est utilisé pour indiquer la raison de la déconnexion. Lorsque le code de résultat strongerAuthRequired est retourné avec ce message, il indique que le serveur a détecté qu'une association de sécurité établie entre le client et le serveur a une défaillance inattendue ou a été compromise.

A la transmission de l'avis de déconnexion, le serveur termine de plein gré la session LDAP comme décrit au paragraphe 5.3.

4.5 Opération Search

L'opération Search *(recherche)* est utilisée pour demander à un serveur de retourner, sous réserve des contrôles d'accès et autres restrictions, un ensemble d'entrées satisfaisant à des critères de recherche complexes. Cela peut être utilisé pour lire des attributs provenant d'une seule entrée, provenant d'entrées immédiatement subordonnées à une entrée particulière, ou de tout un sous-ensemble d'entrées.

4.5.1 Demande Search

La demande Search se définit comme suit :

```
SearchRequest ::= [APPLICATION 3] SEQUENCE {
    baseObject      LDAPDN,
    scope           ENUMERATED {
        baseObject              (0),
        singleLevel             (1),
        wholeSubtree            (2),
        ... },
    derefAliases    ENUMERATED {
        neverDerefAliases       (0),
        derefInSearching        (1),
        derefFindingBaseObj     (2),
        derefAlways             (3) },
    sizeLimit       INTEGER (0 ..  maxInt),
    timeLimit       INTEGER (0 ..  maxInt),
    typesOnly       BOOLEAN,
    filter          Filter,
    attributes      AttributeSelection }

AttributeSelection ::= SEQUENCE OF selector LDAPString
            -- La chaîne LDAPString est restreinte à <attributeSelector> au paragraphe 4.5.1.8

Filter ::= CHOICE {
    et              [0] SET SIZE (1..MAX) OF filter Filter,
    or              [1] SET SIZE (1..MAX) OF filter Filter,
    not             [2] Filter,
    equalityMatch   [3] AttributeValueAssertion,
    substrings      [4] SubstringFilter,
    greaterOrEqual  [5] AttributeValueAssertion,
    lessOrEqual     [6] AttributeValueAssertion,
    present         [7] AttributeDescription,
    approxMatch     [8] AttributeValueAssertion,
    extensibleMatch [9] MatchingRuleAssertion,
    ... }

SubstringFilter ::= SEQUENCE {
    type            AttributeDescription,
    substrings      SEQUENCE SIZE (1..MAX) OF substring CHOICE {
        initial [0] AssertionValue,  -- peut survenir au plus une fois
        any     [1] AssertionValue,
        final   [2] AssertionValue } -- peut survenir au plus une fois
    }

MatchingRuleAssertion ::= SEQUENCE {
    matchingRule    [1] MatchingRuleId OPTIONAL,
    type            [2] AttributeDescription OPTIONAL,
    matchValue      [3] AssertionValue,
```

dnAttributes [4] BOOLEAN DEFAULT FALSE }

Noter qu'une opération du style de "list" de X.500 peut être émulée par le client qui demande une opération singleLevel avec un filtre pour vérifier la présence de l'attribut 'objectClass', et qu'une opération du style "read" de X.500 peut être émulée par une opération baseObject Search avec le même filtre. Un serveur qui fournit une passerelle pour X.500 n'est pas obligé d'utiliser les opérations Read ou List, bien qu'il puisse choisir de le faire, et s'il le fait, il doit fournir la même sémantique que l'opération Search de X.500.

4.5.1.1 SearchRequest.baseObject

C'est le nom de l'entrée d'objet de base (qui peut être la racine) se rapportant à ce que la recherche doit effectuer.

4.5.1.2 SearchRequest.scope

Spécifie le domaine sur lequel la recherche doit être effectuée. La sémantique (telle que décrite dans [X.511]) des valeurs définies de ce champ sont :

baseObject : Le domaine est restreint à l'entrée nommée par baseObject.
singleLevel : Le domaine est restreint aux subordonnés immédiats de l'entrée nommée par baseObject.
wholeSubtree : Le domaine est restreint à l'entrée nommée par baseObject et à tous ses subordonnés.

4.5.1.3 SearchRequest.derefAliases

Indique si les entrées d'alias (telles que définies dans la [RFC4512]) doivent ou non être déréférencées durant les étapes de l'opération Search.

L'action de déréférencer un alias inclut de façon récursive de déréférencer les alias qui se réfèrent aux alias.

Les serveurs DOIVENT détecter les bouclages lors du déréférencement d'alias afin d'empêcher les attaques de déni de service de cette tacks nature.

La sémantique des valeurs définies de ce champ est :
neverDerefAliases : Ne pas déréférencer les alias dans la recherche ou la localisation de l'objet de base de Search.
derefInSearching : Lors de la recherche de subordonnés de l'objet de base, déréférencer tout alias dans le domaine de la recherche. Les objets déréférencés deviennent les points de départ d'autres domaines de recherche où l'opération Search est aussi appliquée. Si le domaine de recherche est wholeSubtree, Search continue dans le ou les sous arbres de tout objet déréférencé.Si le domaine de recherche est singleLevel, la recherche est appliquée à tous les objets déréférencés et n'est pas appliquée à leurs subordonnés. Les serveurs DEVRAIENT éliminer les entrées dupliquées sui apparaissent dues au déréférencement d'alias durant la récherche.
derefFindingBaseObj : Déréférence les alias en localisant l'objet de base de Search, mais pas en recherchant les subordonnés de l'objet de base.
derefAlways : Déréference les alias à la fois en recherchant et en localisant l'objet de base de Search.

4.5.1.4 SearchRequest.sizeLimit

Limite de taille qui restreint le nombre maximum d'entrées à retourner en résultat de Search. Une valeur de zéro dans ce champ indique qu'aucune limite de taille demandée par le client n'est effective pour Search. Les serveurs peuvent aussi mettre en application un nombre maximum d'entrées à retourner.

4.5.1.5 SearchRequest.timeLimit

Limite de temps qui restreint la durée maximale (en secondes) admise pour Search. Une valeur de zéro dans ce champ indique qu'aucune restrictions sur la limite de durée demandée par le client n'est effective pour Search. Les serveurs peuvent aussi mettre en application une limite de durée maximale pour Search.

4.5.1.6 SearchRequest.typesOnly

Indique si les résultats de Search doivent contenir à la fois des descriptions et valeurs d'attribut, ou juste des descriptions d'attribut. Le réglage de ce champ à TRUE provoque le retour des seules descriptions d'attribut (et non des valeurs). L réglage de ce champ à FALSE provoque le retour des descriptions et des valeurs d'attribut.

4.5.1.7 SearchRequest.filter

Filtre qui définit les conditions qui doivent être remplies pour que Search corresponde à une entrée donnée.

Les choix 'and', 'or', et 'not' peuvent être utilisés pour former des combinaisons de filtres. Au moins un élément de filtre DOIT être présent dans un choix 'et' ou 'or'. Les autres se comparent aux valeurs individuelles d'attribut des entrées dans le domaine de Search. (Note pour les développeurs : le filtre 'not' est un exemple d'étiquette de choix dans un module à étiquetage implicite. En BER, ceci est traité comme si l'étiquette était explicite.)

Un serveur DOIT évaluer les filtres conformément à la logique à trois valeurs du paragraphe 7.8.1 de [X.511] (1993). En résumé, un filtre évalue à "TRUE", "FALSE", ou "Undefined". Si le filtrer évalue à TRUE pour une entrée particulière, les attributs de cette entrée sont alors retournés au titre du résultat de Search (sous réserve de toute restriction de contrôle d'accès applicable). Si le filtre évalue à FALSE ou Undefined, l'entrée est ignorée pour Search.

Un filtre du choix "and" est TRUE si tous les filtres dans le SET OF (ensemble de) évaluent à TRUE, FALSE si au moins un filtre est FALSE, et Undefined autrement. Un filtre du choix "or" est FALSE si tous les filtres dans le SET OF évaluent à FALSE, TRUE si au moins un filtre est TRUE, et Undefined autrement. Un filtre du choix 'not' est TRUE si le filtre refusé est FALSE, FALSE si il est TRUE, et Undefined si il est Undefined.

Un élément de filtre évalue à Undefined lorsque le serveur ne serait pas capable de déterminer si la valeur d'assertion correspond à une entrée. Parmi quelques exemples :
- Une description d'attribut dans un filtre equalityMatch, substrings, greaterOrEqual, lessOrEqual, approxMatch, ou extensibleMatch n'est pas reconnue par le serveur.
- Le type d'attribut ne définit pas la règle de correspondance appropriée.
- Un identifiant MatchingRuleId dans extensibleMatch n'est pas reconnu par le serveur ou n'est pas valide pour le type d'attribut.
- Le type de filtrage demandé n'est pas mis en œuvre.
- La valeur d'assertion est invalide.

Par exemple, si un serveur n'a pas reconnu le type d'attribut shoeSize, les filtres (shoeSize=*), (shoeSize=12), (shoeSize>=12), et (shoeSize<=12) vont chacun évaluer à Undefined.

Les serveurs NE DOIVENT PAS retourner des erreurs si les descriptions ou règles de correspondance d'attribut ne sont pas reconnues, si les valeurs d'assertion sont invalides, ou si la syntaxe d'assertion n'est pas prise en charge. Des précisions supplémentaires sur le processus de filtrage figurent au paragraphe 7.8 de [X.511].

4.5.1.7.1SearchRequest.filter.equalityMatch
La règle de correspondance pour un filtre equalityMatch est définie par la règle de correspondance EQUALITY pour le type ou sous-type d'attribut. Le filtre est TRUE lorsque la règle EQUALITY retourne TRUE appliqué à l'attribut ou sous-type et la valeur affirmée.

4.5.1.7.2SearchRequest.filter.substrings
Il DOIT y avoir au plus un 'initial' et au plus un 'final' dans le 'substrings' d'un SubstringFilter. Si 'initial' est présent, il DOIT être le premier élément de 'substrings'. Si 'final' est présent, il DOIT être le dernier élément de 'substrings'.

La règle de correspondance pour une AssertionValue dans un élément de filtre substrings est définie par la règle de correspondance SUBSTR pour le type ou sous-type d'attribut. Le filtre est TRUE lorsque la règle SUBSTR retourne TRUE appliqué à l'attribut ou sous-type et la valeur affirmée.

Noter que l'AssertionValue dans un élément de filtre substrings se conforme à la syntaxe d'assertion de la règle de correspondance EQUALITY pour le type d'attribut plutôt qu'à la syntaxe d'assertion de la règle de correspondance SUBSTR pour le type d'attribut. Par nature, le SubstringFilter tout ebntier est converti en une valeur d'assertion de la règle de correspondance substrings avant d'appliquer la règle.

4.5.1.7.3SearchRequest.filter.greaterOrEqual
La règle de correspondance pour un filtre greaterOrEqual est définie par la règle de correspondance ORDERING pour le type ou sous-type d'attribut. Le filtre est TRUE lorsque la règle ORDERING retourne FALSE appliqué à l'attribut ou sous-type et la valeur affirmée.

4.5.1.7.4SearchRequest.filter.lessOrEqual

Les règles de correspondance pour un filtre lessOrEqual sont définies par les règles de correspondance ORDERING et EQUALITY pour le type ou sous-type d'attribut. Le filtre est TRUE lorsque la règle ORDERING ou EQUALITY retourne TRUE appliqué à l'attribut ou au sous-type et à la valeur affirmée.

4.5.1.7.5SearchRequest.filter.present

Un filtre present est TRUE lorsqu'il y a un attribut ou sous-type de la description d'attribut spécifié présente dans une entrée, FALSE lorsque aucun attribut or sous-type de la description d'attribut spécifié n'est présent dans une entrée, et Undefined autrement.

4.5.1.7.6SearchRequest.filter.approxMatch

Un filtre approxMatch est TRUE lorsque il y a une valeur du type ou sous-type d'attribut pour lequel un algorithme de correspondance approximative défini localement (par exemple, variations orthographiques, correspondance phonétique, etc.) retourne TRUE. Si une valeur correspond en égalité, elle satisfait aussi à une correspondance approximative. Si une correspondance approximative n'est pas acceptée pour l'attribut, cet élément de filtre devrait être traité comme un equalityMatch.

4.5.1.7.7SearchRequest.filter.extensibleMatch

Les champs de l'élément de filtre extensibleMatch sont évalués comme suit :

- Si le champ matchingRule est absent, le champ de type DOIT être présent, et une confrontation d'égalité est effectuée pour ce type.
- Si le champ type field est absent et si matchingRule est présent, la valeur matchValue est comparée à tous les attributs dans une entrée qui accepte cette matchingRule.
- Si le champ type est présent et si matchingRule est présent, la valeur matchValue est comparée au type d'attribut spécifié et ses sous-types.
- Si le champ dnAttributes est mis à TRUE, la confrontation est de plus appliquée à toutes les AttributeValueAssertions de nom distinctif dans le nom distinctif d'une entrée, et elle évalue à TRUE si il y a au moins un attribut ou sous-type dans le nom distinctif pour lequel l'élément de filtre évalue à TRUE. Le champ dnAttributes est présent pour atténuer le besoin de plusieurs versions de règles de correspondance génériques (telles qu'une correspondance des mots), où l'une s'applique aux entrées et un autre s'applique aussi bien aux attributs d'entrées et de noms distinctifs.

La règle matchingRule utilisée pour l'évaluation détermine la syntaxe pour la valeur de l'assertion. Une fois que la règle matchingRule et le ou les attributes ont été déterminés, l'élément de filtre évalue à TRUE si il correspond au moins à un type ou sous-type d'attribut dans l'entrée, FALSE si il ne correspond à aucun type d'attribut ou sous-type dans l'entrée, et Undefined si la matchingRule n'est pas reconnue, si la matchingRule ne convient pas pour l'usage avec le type spécifié, ou si la valeur assertionValue est invalide.

4.5.1.8 SearchRequest.attributes

C'est une liste de sélection des attributs à retourner de chaque entrée qui satisfait au filtre de recherche. Les attributs qui sont des sous-types des attributs figurant sur la liste sont implicitement inclus. Les valeurs LDAPString de ce champ sont restreintes aux formes Backus-Naur augmenté (ABNF) [RFC4234] suivantes :

attributeSelector = attributedescription / selectorspecial

selectorspecial = noattrs / alluserattrs

noattrs = %x31.2E.31 ; "1.1"

alluserattrs = %x2A ; asterisk ("*")

La production de <attributedescription> est définie au paragraphe 2.5 de la [RFC4512].

Il y a trois cas particuliers qui peuvent apparaître dans la liste de sélection des attributs :

1. Une liste vide sans attribut demande de retourner tous les attributs d'utilisateur.
2. Une liste contenant "*" (avec zéro, une ou plusieurs descriptions d'attribut) demande de retourner tous les attributs d'utilisateur en plus des autres attributs (opérationnels) listés.
3. Une liste ne contenant que l'OID "1.1" indique qu'aucun attribut n'est à retourner. Si "1.1" est fourni avec d'autres valeurs de attributeSelector, le attributeSelector "1.1" sera ignoré. Cet OID a été choisi parce qu'il ne correspond à aucun attribut utilisé.

Les mises en œuvre de client devraient noter que même si tous les attributs d'utilisateur sont demandés, certains attributs et/ou valeurs d'attribut de l'entrée peuvent n'être pas inclus dans les résultats de Search du fait des contrôles d'accès ou autres restrictions. De plus, les serveurs ne vont pas retourner d'attributs opérationnels, tels que objectClasses ou attributeTypes, s'ils ne sont pas listés par nom. Les attributs opérationnels sont décrits dans la [RFC4512].

Les attributs sont retournés au plus une fois dans une entrée. Si une description d'attribut est nommée plus d'une fois dans la liste, les noms suivants sont ignorés. Si une description d'attribut dans la liste n'est pas reconnue, elle est ignorée par le serveur.

4.5.2 Résultat de Search

Les résultats de l'opération Search sont retournés comme zéro, un ou plusieurs messages SearchResultEntry et/ou SearchResultReference, suivis par un seul message SearchResultDone.

```
SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
    objectName      LDAPDN,
    attributes      PartialAttributeList }

PartialAttributeList ::= SEQUENCE OF
                partialAttribute PartialAttribute

SearchResultReference ::= [APPLICATION 19] SEQUENCE
                    SIZE (1..MAX) OF uri URI

SearchResultDone ::= [APPLICATION 5] LDAPResult
```

Chaque SearchResultEntry représente une entrée trouvée durant Search. Chaque SearchResultReference représente une zone non encore explorée durant le Search. Les messages SearchResultEntry et SearchResultReference peuvent venir dans n'importe quel ordre. A la suite de toutes les réponses SearchResultReference et SearchResultEntry, le serveur retourne une réponse SearchResultDone, qui contient une indication de succès ou le détail de toute erreur qui serait survenue.

Chaque entrée retournée dans une SearchResultEntry contiendra tous les attributs appropriés, comme spécifié dans le champ d'attributs de la demande Search, sous réserve des contrôles d'accès et autres règles administratives. Noter que PartialAttributeList peut contenir zéro élément.

Ceci peut arriver lorsque aucun des attributs d'une entrée n'était demandé ou n'a pu être retourné. Noter aussi que partialAttribute vals set peut ne contenir aucun élément. Ceci peut arriver lorsque typesOnly est demandé, si les contrôles d'accès empêchent le retour des valeurs, ou d'autres raisons.

Certains attributs peuvent être construits par le serveur et apparaître dans une liste d'attributs SearchResultEntry, bien qu'ils ne soient pas des attributs mémorisés d'une entrée. Les clients NE DEVRAIENT PAS supposer que tous les attributs peuvent être modifiés, même si c'est permis par le contrôle d'accès.

Si le schéma du serveur définit des noms abrégés [RFC4512] pour un type d'attribut, le serveur DEVRAIT alors utiliser un de ces noms dans les descriptions d'attribut pour ce type d'attribut (de préférence à l'utilisation du format <numericoid> [RFC4512] de l'identifiant d'objet du type d'attribut). Le serveur NE DEVRAIT PAS utiliser le nom abrégé si ce nom est connu comme ambigu par le serveur, ou si il doit autrement causer vraisemblablement des problèmes d'interopérabilité.

4.5.3 Références de continuation dans le résultat de Search

Si le serveur a été capable de localiser l'entrée à laquelle se réfère le baseObject mais n'a pas été capable ou ne veut pas rechercher une ou plusieurs entrées non locales, le serveur peut retourner un ou plusieurs messages SearchResultReference, contenant chacun une référence à un autre ensemble de serveurs pour continuer l'opération. Un serveur NE DOIT PAS retourner de message SearchResultReference si il n'a pas localisé le baseObject et donc n'a pas recherché d'entrées. Dans ce cas, il devrait retourner un SearchResultDone contenant un renvoi de référence ou un code de résultat noSuchObject (selon ce que le serveur sait de l'entrée nommée dans le baseObject).

Si un serveur détient une copie ou une copie partielle du contexte de dénomination subordonné (Section 5 de la [RFC4512]), il peut utiliser le filtre de recherche pour déterminer s'il doit ou non retourner une réponse SearchResultReference. Autrement, les réponses SearchResultReference sont toujours retournées lorsqu'elles sont dans le domaine.

SearchResultReference est du même type de données que Referral.

Si le client souhaite continuer Search, il produit une nouvelle opération Search pour chaque SearchResultReference retourné. Si plusieurs URI sont présents, le client supposera que tout URI accepté peut être utilisé pour continuer l'opération.

Les clients qui suivent les références de continuation de recherche DOIVENT s'assurer qu'ils ne font pas des boucles entre les serveurs. Ils NE DOIVENT PAS contacter de façon répétée le même serveur pour la même demande avec les mêmes paramètres. Certains clients utilisent un compteur qui est incrémenté à chaque fois que survient un traitement de résultat de résultat de recherche pour une opération, et cette sorte de client DOIT être capable de traiter au moins dix renvois de références enchâssés lors du traitement de l'opération.

Noter que l'opération Abandon décrite au paragraphe 4.11 ne s'applique qu'à l'opération particilière envoyée à la couche de message LDAP entre un client et un serveur. Le client doit abandonner individuellement les opérations Search qu'il souhaite.

Un URI pour un serveur mettant en œuvre LDAP et accessible via TCP/IP (v4 ou v6) [RFC793][RFC791] est écrit comme un URL LDAP conformément à la [RFC4516].

Les valeurs de SearchResultReference qui sont des URL LDAP suivent les règles ci-après :

- La partie <dn> de l'URL LDAP DOIT être présente, avec le nouveau nom d'objet cible. Le client utilise ce nom en suivant la référence.
- Certains serveurs (par exemple, qui participent à une indexation distribuée) peuvent fournir un filtre différent dans l'URL LDAP.
- Si la partie <filter> de l'URL LDAP est présente, le client utilise ce filtre dans sa demande suivante pour continuer cette Search, et elle n'est pas présente, le client utilise le même filtre qu'il a utilisé pour cette Search.
- Si le domaine de recherche d'origine était singleLevel, la partie <scope> de l'URL LDAP sera "base".
- Il est RECOMMANDÉ que la partie <scope> soit présente pour éviter les ambiguïtés. En l'absence d'une partie <scope>, le domaine de la demande Search d'origine sera supposé.
- D'autres aspects de la nouvelle demande Search peuvent être les mêmes ou être différents de ceux de la demande Search qui a générée la SearchResultReference.
- Le nom d'un sous-arbre non exploré dans une SearchResultReference ne doit pas nécessairement être subordonné à l'objet de base.

D'autres sortes d'URI peuvent être retournées. La syntaxe et la sémantique de tels URI fera l'objet de spécifications à l'avenir. Les clients peuvent ignorer les URI qu'ils ne prennent pas en charge.

Les caractères codés en UTF-8 qui apparaissent dans la représentation de chaîne d'un nom distinctif, d'un filtre de recherche, ou dans d'autres champs de la valeur du renvoi de références peuvent n'être pas légaux pour les URI (par exemple, des espaces) et DOIVENT être convertis en utilisant la méthode du pourcentage (%) de la [RFC3986].

4.5.3.1 Exemples

Par exemple, supposons que le serveur contacté (hosta) détient l'entrée <DC=Example,DC=NET> et l'entrée <CN=Manager,DC=Example,DC=NET>. Il sait que les deux serveurs LDAP (hostb) et (hostc) détiennent <OU=People,DC=Example,DC=NET> (l'un est le maître et l'autre serveur est son ombre), et qu'un serveur à capacité LDAP (hostd) détient le sous-arbre <OU=Roles,DC=Example,DC=NET>. Si une recherche Search wholeSubtree de <DC=Example,DC=NET> est demandée au serveur contacté, il peut retourner ce qui suit :

```
SearchResultEntry for DC=Example,DC=NET
SearchResultEntry for CN=Manager,DC=Example,DC=NET
SearchResultReference {
```

```
  ldap://hostb/OU=People,DC=Example,DC=NET??sub
  ldap://hostc/OU=People,DC=Example,DC=NET??sub }
SearchResultReference {
  ldap://hostd/OU=Roles,DC=Example,DC=NET??sub }
SearchResultDone (success)
```

Les mises en œuvre de clients devraient noter qu'en suivant une SearchResultReference, des SearchResultReference supplémentaires peuvent être générées. En continuant l'exemple, si le client a contacté le serveur (hostb) et produit la demande Search pour le sous-arbre <OU=People,DC=Example,DC=NET>, le serveur pourrait répondre comme suit :

```
SearchResultEntry for OU=People,DC=Example,DC=NET
SearchResultReference {
  ldap://hoste/OU=Managers,OU=People,DC=Example,DC=NET??sub }
SearchResultReference {
  ldap://hostf/OU=Consultants,OU=People,DC=Example,DC=NET??sub }
SearchResultDone (success)
```

De même, si un Search singleLevel de <DC=Example,DC=NET> est demandé au serveur contacté, il peut retourner ce qui suit :

```
SearchResultEntry for CN=Manager,DC=Example,DC=NET
SearchResultReference {
  ldap://hostb/OU=People,DC=Example,DC=NET??base
  ldap://hostc/OU=People,DC=Example,DC=NET??base }
SearchResultReference {
  ldap://hostd/OU=Roles,DC=Example,DC=NET??base }
SearchResultDone (success)
```

Si le serveur contacté ne détient pas l'objet de base pour la recherche, mais a connaissance de sa localisation possible, il peut alors retourner un renvoi de références au client. Dans ce cas, si le client demande une recherche de sous-arbre de <DC=Example,DC=ORG> à hosta, le serveur retourne un SearchResultDone contenant un renvoi de référence.

```
SearchResultDone (referral) {
  ldap://hostg/DC=Example,DC=ORG??sub }
```

4.6 Opération Modify

L'opération Modify permet à un client de demander que soit effectuée en son nom la modification d'une entrée par un serveur. La demande Modify est définie comme suit :

```
ModifyRequest ::= [APPLICATION 6] SEQUENCE {
    object          LDAPDN,
    changes         SEQUENCE OF change SEQUENCE {
        operation       ENUMERATED {
            add     (0),
            delete  (1),
            replace (2),
            ...  },
        modification    PartialAttribute } }
```

Les champs de la demande Modify sont :

- object : La valeur de ce champ contient le nom de l'entrée à modifier. Le serveur NE DOIT PAS effectuer de déréférencement d'alias en déterminant l'objet à modifier.
- changes : C'est une liste des modifications à effectuer sur l'entrée. La liste entière des modifications DOIT être effectuée dans l'ordre de la liste comme une seule opération atomique. Alors que des modifications individuelles peuvent violer certains aspects du schéma de répertoire (comme la règle de contenu de la définition de classe d'objet et de l'arbre d'information de répertoire (DIT, *Directory Information Tree*)), l'entrée résultante après avoir

effectué la liste entière des modifications DOIT se conformer aux exigence du modèle de répertoire model et au schéma de contrôle [RFC4512].

- operation : Utilisé pour spécifier le type de modification effectuée. Chaque type d'opération agit sur la modification suivante. Les valeurs de ce champ ont respectivement la sémantique suivante :
  - add : ajoute les valeurs citées à l'attribut de modification, créant l'attribut si nécessaire.
  - delete : supprime les valeurs citées de l'attribut de modification. Si aucune valeur n'est citée, ou si toutes les valeurs actuelles de l'attribut sont citées, l'attribut entier est supprimé.
  - replace : remplace toutes les valeurs existantes de l'attribut de modification avec les nouvelles valeurs citées, créant l'attribut si il n'existe pas déjà. Un replace sans valeur va supprimer l'attribut entier s'il existe, et il est ignoré si l'attribut n'existe pas.
- modification : Attribut partiel (qui peut avoir un SET of vals *(ensemble de valeurs)* vide) utilisés pour détenir le type d'attribut ou type d'attribut et valeurs modifié.

A réception d'une demande Modify, le serveur essaye d'effectuer les modifications nécessaires au DIT et retourne le résultat dans une réponse Modify, définie comme suit :

ModifyResponse ::= [APPLICATION 7] LDAPResult

Le serveur va retourner au client une seule réponse Modify indiquant l'achèvement réussi de la modification DIT, ou la raison de l'échec de la modification. Du fait des exigences d'atomicité dans l'application de la liste des modifications dans la demande Modify, le client peut s'attendre à ce qu'aucune modifications du DIT n'ait été effectuée si la réponse Modify reçue indique une erreur de quelque sorte, et que toutes les modifications demandées aient été effectuées si la réponse Modify indique l'achèvement réussi de l'opération Modify. Le client ne peut pas déterminer si la modification a été ou non appliquée si la réponse Modify n'a pas été reçue (par exemple, la session LDAP s'est terminée ou l'opération Modify a été abandonnée).

Les serveurs DOIVENT s'asurer que les entrées se conforment aux règles de schéma d'utilisateur et de système ou autres contraintes de modèle de données. L'opération Modify ne peut pas être utilisée pour retirer d'une entrée une de ses valeurs distinctives, c'est-à-dire, de ces valeurs qui forment le nom distinctif relatif de l'entrée. Toute tentative de la faire résultera dans le retour par le serveur du code de résultat notAllowedOnRDN. L'opération ModifyDN décrite au paragraphe 4.9 est utilisée pour renommer une entrée.

Pour les types d'attribut qui ne spécifient pas de confrontation d'égalité, on suit les règles du paragraphe 2.5.1 de la [RFC4512].

Noter que du fait des simplifications apportées à LDAP, il n'y a plus de transposition directe entre les changements dans une ModifyRequest de LDAP et les changements d'une opération ModifyEntry DAP, et des mises en œuvre différentes de passerelles LDAP-DAP pourront utiliser des moyens différents pour représenter le changement. Pour être réussi, l'effet final des opérations sur l'entrée DOIT être identique.

4.7 Opération Add

L'opération Add permet à un client de demander l'ajout d'une entrée dans un répertoire. La demande Add est définie comme suit :

```
AddRequest ::= [APPLICATION 8] SEQUENCE {
    entry        LDAPDN,
    attributes     AttributeList }

AttributeList ::= SEQUENCE OF attribute Attribute
```

Les champs de la demande Add sont :

- entry : nom de l'entrée à ajouter. Le serveur NE DOIT PAS déréférencer d'alias en localisant l'entrée à ajouter.

- attributes : liste des attributs qui, avec ceux du RDN, constituent le contenu de l'entrée à ajouter. Les clients PEUVENT ou PEUVENT NE PAS inclure le ou les attributs de RDN dans cette liste. Les clients NE DOIVENT PAS fournir d'attributs NO-USER-MODIFICATION tels que les attributs createTimestamp ou creatorsName attributes, car le serveur les maintient automatiquement.

Les serveurs DOIVENT s'assurer que les entrées se conforment aux règles de schéma d'utilisateur et de système ou autres contraintes de modèle de données. Pour les types d'attribut qui ne spécifient pas de confrontation d'égalité, on suit les règles du paragraphe 2.5.1 de la [RFC4512] (ceci s'applique à l'attribut de dénomination en plus de l'ajout de tous attributs multi-valeurs).

L'entrée nommée dans le champ d'entrée de AddRequest NE DOIT PAS exister pour que la demande AddRequest réussisse. Le supérieur immédiat (parent) d'une entrée d'objet ou d'alias à ajouter DOIT exister. Par exemple, si le client essaye d'ajouter <CN=JS,DC=Example,DC=NET>, si l'entrée <DC=Example,DC=NET> n'existe pas, et si l'entrée <DC=NET> existe, le serveur retournera alors le code de résultat noSuchObject avec le champ matchedDN contenant <DC=NET>.

A réception d'une demande Add, un serveur essayera d'ajouter l'entrée demandée. Le résultat de la tentative Add sera retourné au client dans la réponse Add, définie comme suit :

```
AddResponse ::= [APPLICATION 9] LDAPResult
```

Une réponse de succès indique que la nouvelle entrée a été ajoutée au répertoire.

4.8 Opération Delete

L'opération Delete permet à un client de demander le retrait d'une entrée du répertoire. La demande Delete est définie comme suit :

```
DelRequest ::= [APPLICATION 10] LDAPDN
```

La demande Delete comporte le nom de l'entrée à supprimer. Le serveur NE DOIT PAS déréférencer les alias lors de la résolution du nom de l'entrée cible à retirer.

Seules les entrées non ramifiées (celles qui n'ont pas d'entrées subordonnées) peuvent être supprimées par cette opération.

A réception d'une demande Delete, un serveur essayera d'effectuer la suppression de l'entrée demandée et retournera le résultat dans la réponse Delete définie comme suit :

```
DelResponse ::= [APPLICATION 11] LDAPResult
```

4.9 Opération ModifyDN

L'opération ModifyDN permet à un client de changer le nom distinctif relatif (RDN, *Relative Distinguished Name*) d'une entrée dans le répertoire et/ou de déplacer un sous-arbre d'entrées à une nouvelle localisation dans le répertoire. La demande ModifyDN est définie comme suit :

```
ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
    entry          LDAPDN,
    newrdn         RelativeLDAPDN,
    deleteoldrdn   BOOLEAN,
    newSuperior    [0] LDAPDN OPTIONAL }
```

Les champs de la demande ModifyDN sont :

- entry : nom de l'entrée à changer. Cette entrée peut avoir des entrées subordonnées ou pas.

- newrdn : nouveau RDN de l'entrée. La valeur de l'ancien RDN est fournie lorsqu'on déplace l'entrée sur un nouveau supérieur sans changer son RDN. Les valeurs d'attribut du nouveau RDN qui ne correspondent à aucune valeur d'attribut de l'entrée sont ajoutées à l'entrée, et une erreur appropriée est retournée si cela échoue.
- deleteoldrdn : champ booléen qui contrôle si les valeurs d'attribut de l'ancien RDN sont à conserver comme attributs de l'entrée ou à supprimer de l'entrée.
- newSuperior : s'il est présent, c'est le nom d'une entrée d'objet existant qui devient le supérieur immédiat (parent) de l'entrée existante.

Le serveur NE DOIT PAS déréférencer d'alias en localisant les objets nommés dans entry ou newSuperior.

A réception d'une ModifyDNRequest, un serveur essayera d'effectuer le changement de nom et de retourner le résultat dans la réponse ModifyDN, définie comme suit :

ModifyDNResponse ::= [APPLICATION 13] LDAPResult

Par exemple, si l'entrée nommée dans le champ entry était <cn=John Smith,c=US>, si le champ newrdn était <cn=John Cougar Smith>, et si le champ newSuperior était absent, cette opération essayerait alors de renommer l'entrée <cn=John Cougar Smith,c=US>. Si il y avait déjà une entrée portant ce nom, l'opération échouerait avec le code de résultat entryAlreadyExists.

Les servers DOIVENT s'assurer que les entrées se conforment aux règles de schéma d'utilisateur et de système ou autres contraintes de modèle de données. Pour les types d'attribut qui ne spécifient par de confrontation d'égalité, on suit les règles du paragraphe 2.5.1 de la [RFC4512] (ceci se rapporte à newrdn et deleteoldrdn).

L'objet nommé dans newSuperior DOIT exister. Par exemple, si le client a essayé d'ajouter <CN=JS,DC=Example,DC=NET>, si l'entrée <DC=Example,DC=NET> n'existe pas, et si l'entrée <DC=NET> existe, le serveur retournerait alors le code de résultat noSuchObject avec le champ matchedDN contenant <DC=NET>.

Si le champ deleteoldrdn est TRUE, les valeurs d'attribut qui forment le RDN ancien (mais pas le nouveau RDN) sont supprimées de l'entrée. Si le champ deleteoldrdn est FALSE, les valeurs d'attribut qui forment le RDN ancien seront conservées comme valeurs d'attribut non distinctif de l'entrée.

Noter que X.500 restreint l'opération ModifyDN pour n'affecter que les entrées qui sont contenues dans un seul serveur. Si le serveur LDAP est transposé en DAP, cette restriction ne doit alors pas s'applique, et le code de résultat affectsMultipleDSAs sera retourné si cette erreur survient. En général, les clients NE DOIVENT PAS s'attendre à être capables d'effectuer des mouvements arbitraires des entrées et des arborescences entre les serveurs ou entre contextes de dénomination

4.10 Opération Compare

L'opération Compare permet à un client de comparer une valeur d'assertion avec la valeurs d'un attribut particulier dans une entrée particulière du répertoire. La demande Compare est définie comme suit :

```
CompareRequest ::= [APPLICATION 14] SEQUENCE {
    entry       LDAPDN,
    ava         AttributeValueAssertion }
```

Les champs de la demande Compare sont :

- entry : nom de l'entrée à comparer. Le serveur NE DOIT PAS déréférencer d'alias en localisant l'entrée à comparer.
- ava : détient l'assertion de valeur d'attribut à comparer.

A réception d'une demande Compare, un serveur va essayer d'effectuer la comparaison demandée et de retourner le résultat dans la réponse Compare, définie comme suit :

CompareResponse ::= [APPLICATION 15] LDAPResult

Le code de résultat est réglé à compareTrue, compareFalse, ou à une erreur appropriée. compareTrue indique que la valeur de l'assertion dans le champ ava correspond à une valeur de l'attribut ou du sous-type conforme à la règle de correspondance EQUALITY de l'attribute. compareFalse indique que la valeur de l'assertion dans le champ ava et les valeurs de l'attribut ou sous-type ne correspondaient pas. Les autres codes de résultat indiquent que le résultat de la comparaison était Undefined (paragraphe 4.5.1.7), ou qu'une autre erreur est survenue.

Noter que certains systèmes de répertoire peuvent établir des contrôles d'accès qui permettent de comparer les valeurs de certains attributs (comme un userPassword) mais pas de les interroger par d'autres moyens.

4.11 Opération Abandon

La fonction de l'opération Abandon est de permettre à un client de demander que le serveur abandonne une opération non achevée. La demande Abandon est définie comme suit :

AbandonRequest ::= [APPLICATION 16] MessageID

Le MessageID est celui d'une opération qui a été demandé précédemment à cette couche de message LDAP. La demande Abandon elle-même a son propre MessageID. Il est distinct du MessageID de l'opération précédente en cours d'abandon.

Il n'y a pas de réponse définie dans l'opération Abandon. A réception d'une AbandonRequest, le serveur PEUT abandonner l'opération identifiée par le MessageID. Comme le client ne peut par dire la différence entre une opération dont l'abandon est réussi et une opération non achevée, l'application de l'opération Abandon est limitée aux utilisations où le client ne demande pas d'indication de son résultat.

Les opérations Abandon, Bind, Unbind, et StartTLS ne peuvent pas être abandonnées.

Dans le cas où un serveur reçoit une demande Abandon sur une opération Search pendnat la transmission des réponses à Search, ce serveur DOIT cesser immédiatement de transmettre des réponses d'entrée à la demande abandonnée, et il NE DOIT PAS envoyer le SearchResultDone. Bien sûr, le serveur DOIT s'assurer que seuls sont transmis des PDU de message LDAP codés de façon appropriée.

La capacité à abandonner d'autres opérations (en particulier de mise à jour) est à la discrétion du serveur.

Les clients ne devraient pas envoyer de demandes Abandon pour la même opération plusieurs fois, et ils DOIVENT aussi être prêts à recevoir des résultats d'opérations qu'ils ont abandonnées (car elles peuvent avoir été en transit lors de la demande Abandon ou peuvent n'être pas capable d'être abandonnées).

Les serveurs DOIVENT éliminer les demandes Abandon pour les messageID qu'ils ne reconnaissent pas, pour les opérations qu'ils ne peuvent pas abandonner, et pour les opérations qu'ils ont déjà abandonnées.

4.12 Opération Extended

L'opération Extended permet de définir des opérations supplémentaires pour des services qui ne sont pas disponibles dans le protocole ; par exemple, des opérations Add pour installer la sécurité de couche transport (paragraphe 4.14).

L'opération Extended permet aux clients de faire des demandes et recevoir des réponses avec des syntaxes et sémantiques prédéfinies. Celle-ci peuvent être définies dans des RFC ou être pour des mises en œuvre particulières.

Chaque opération Extended consiste en une demande Extended et une réponse Extended.

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
    requestName      [0] LDAPOID,
    requestValue     [1] OCTET STRING OPTIONAL }
```

Le requestName est une représentation décimale à octets séparés par des ponts de l'IDENTIFIANT D'OBJET unique correspondant à la demande. La requestValue est une information sous une forme définie par cette demande, encapsulée dans une CHAINE D'OCTETS.

Le serveur va répondre par un LDAPMessage contenant une ExtendedResponse.

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
    COMPONENTS OF LDAPResult,
    responseName     [10] LDAPOID OPTIONAL,
    responseValue    [11] OCTET STRING OPTIONAL }
```

Le champ responseName, lorsqu'il est présent, contient un LDAPOID qui est unique pour cette opération étendue ou réponse. Ce champ est facultatif (même lorsque la spécification d'extension définit un LDAPOID à utiliser dans ce champ). Le champ sera absent chaque fois que le serveur est incapable de déterminer le LDAPOID approprié à retourner ou qu'il ne veut pas le faire, par exemple, lorsque le requestName ne peut pas être analysé ou que sa valeur n'est pas reconnue.

Lorsque le requestName n'est pas reconnu, le serveur retourne une protocolError. (Le serveur peut retourner protocolError dans d'autres cas.)

Les champs requestValue et responseValue contiennent des informations associées à l'opération. Le format de ces champs est défini par la spécification de l'opération Extended. Les mises en œuvre DOIVENT être prêtes à traiter des contenus arbitraires de ces champs, y compris des octets à zéro. Les valeurs qui sont définies en termes d'ASN.1 et de codage en BER, conformément au paragraphe 5.1 suivent aussi les règles d'extensibilité de la Section 4.

Les serveurs font la liste des requestName des demandes Extended qu'ils reconnaissent dans l'attribut 'supportedExtension' dans le DSE racine (paragraphe 5.1 de la [RFC4512]).

Les opérations Extended peuvent être spécifiées dans d'autres documents. La spécification d'une opération Extended consiste en :

- l'IDENTIFIANT D'OBJET alloué au requestName,
- l'IDENTIFIANT D'OBJET (s'il en est) alloué au responseName (noter que le même IDENTIFIANT D'OBJET peut être utilisé pour requestName et responseName),
- le format du contenu de la requestValue et responseValue (s'il en est), et
- la sémantique de l'opération.

4.13 Message IntermediateResponse

Alors que l'opération Search procure un mécanisme pour retourner plusieurs messages de réponse pour une seule demande Search, d'autres opérations, par nature, ne fournissent pas plusieurs messages de réponse.

Le message IntermediateResponse fournit un mécanisme général de définition d'opérations demande unique/réponses multiples dans LDAP. Ce message est destiné à une utilisation conjointe avec l'opération Extended pour définir de nouvelles opérations demande unique/réponses multiples ou conjointement avec un contrôle lorsque l'extension d'opérations LDAP existantes les oblige à retourner des informations de réponse Intermediate.

Il est prévu que les définitions et descriptions des opérations et contrôles Extended qui font usage du message IntermediateResponse définissent les circonstances dans lesquelles un message IntermediateResponse peut être envoyé par un serveur et la signification associée à un message IntermediateResponse envoyé dans des circonstances particulières.

```
IntermediateResponse ::= [APPLICATION 25] SEQUENCE {
      responseName     [0] LDAPOID OPTIONAL,
      responseValue    [1] OCTET STRING OPTIONAL }
```

Les messages IntermediateResponse NE DOIVENT PAS être retournés au client sauf si le client produit une demande qui sollicite spécifiquement de les retourner. Le présent document définit deux formes de sollicitation : l'opération Extended et le contrôle de demande. Les messages IntermediateResponse sont spécifiés dans des documents qui décrivent la manière dont ils sont sollicités (c'est-à-dire, dans la spécification d'opération Extended ou de contrôle de demande qui les utilise). Ces spécifications incluent :

- l'IDENTIFIANT D'OBJET (s'il en est) alloué au responseName,
- le format du contenu de la responseValue (s'il en est), et
- la sémantique associée au message IntermediateResponse.

Les extensions qui permettent de retourner plusieurs types de messages IntermediateResponse DOIVENT identifier les types qui utilisent des valeurs uniques de responseName (noter qu'une d'elles peut ne spécifier aucune valeur).

Les paragraphes 4.13.1 et 4.13.2 décrivent des exigences supplémentaires sur l'inclusion de responseName et responseValue dans les messages IntermediateResponse.

4.13.1 Usage avec ExtendedRequest et ExtendedResponse de LDAP

Une opération demande unique/réponses multiples peut être définie en utilisant un seul message ExtendedRequest pour solliciter zéro, un ou plusieurs messages IntermediateResponse d'une ou plusieurs sortes, suivis par un message ExtendedResponse.

4.13.2 Usage avec contrôle de demande de LDAP

Une sémantique de contrôle peut inclure de retourner zéro, un ou plusieurs messages IntermediateResponse avant de retourner le code de résultat final pour l'opération. Une ou plusieurs sortes de messages IntermediateResponse peuvent être envoyés en réponse à une demande control.

Tous les messages IntermediateResponse associés à des contrôles de demande DOIVENT inclure un responseName. Cette exigence garantit que le client peut correctement identifier la source des messages IntermediateResponse lorsque :

- deux contrôles ou plus utilisant des messages IntermediateResponse sont inclus dans une demande d'opération LDAP quelconque ou
- un ou plusieurs contrôles utilisant des messages IntermediateResponse sont inclus dans une demande avec une opération Extended de LDAP qui utilise de smessages IntermediateResponse.

4.14 Opération StartTLS

L'objet de l'opération de lancement de la sécurité de couche transport (StartTLS) est d'initialiser l'installation d'une couche TLS. L'opération StartTLS est définie en utilisant le mécanisme d'opération Extended décrit au paragraphe 4.12.

4.14.1 Demande StartTLS

Un client demande l'établissement de TLS en transmettant un message de demande StartTLS au serveur. La demande StartTLS est définie en termes de ExtendedRequest. Le requestName est "1.3.6.1.4.1.1466.20037", et le champ requestValue est toujours absent.

Le client NE DOIT PAS envoyer de PDU LDAP à cette couche de message LDAP à la suite de cette demande jusqu'à ce qu'il ait reçu une réponse Extended StartTLS et, dans le cas d'une réponse de succès, qu'il achève les négociations TLS.

Des problèmes de séquencement détectés (en particulier ceux détaillés au paragraphe 3.1.1 de la [RFC4513]) ont pour résultat le réglage du resultCode à operationsError.

Si le serveur ne prend pas en charge TLS (par conception ou par la configuration en cours), il retourne le code de résultat réglé à protocolError comme décrit au paragraphe 4.12.

4.14.2 StartTLS Response

Lorsqu'une demande StartTLS est reçue, les serveurs qui prennent en charge l'opération DOIVENT retourner un message de réponse StartTLS au demandeur. Le responseName est "1.3.6.1.4.1.1466.20037" lorsqu'il est fourni (voir au paragraphe 4.12). La valeur responseValue est toujours absente.

Si le serveur veur et est capable de négocier TLS, il retourne la réponse StartTLS avec le code de résultat réglé à succès. A réception par le client d'une réponse StartTLS de succès, les homologues de protocole peuvent commencer la négociation TLS comme exposé à la Section 3 de la [RFC4513].

Si par ailleurs, le serveur ne veut pas ou ne peut pas effectuer cette opération, le serveur va retourner un code de résultat approprié indiquant la nature du problème. Par exemple, si le sous-système TLS n'est pas actuellement disponible, le serveur peut l'indiquer en retournant le code de résultat réglé à indisponible. Dans les cas où un code de résultat de non réussite est retourné, la session LDAP est laissée sans couche TLS.

4.14.3 Retrait de la couche TLS

Le client ou le serveur PEUVENT retirer la couche TLS et laisser la couche de message LDAP intacte en envoyant et recevant une alerte de clôture TLS.

L'homologue de protocole initiateur envoie l'alerte de clôture TLS et DOIT attendre jusqu'à ce qu'il reçoive une alerte de clôture TLS de l'autre homologue avant d'envoyer d'autres PDU LDAP.

Lorsqu'un homologue de protocole reçoit l'alerte de clôture TLS initiale, il peut choisir de permettre à la couche de message LDAP de rester intacte. Dans ce cas, il DOIT immédiatement transmettre une alerte de clôture TLS. Après cela, il PEUT envoyer et recevoir des PDU LDAP.

Les homologues de protocole PEUVENT terminer la session LDAP après l'envoi ou la réception d'une alerte de clôture TLS.

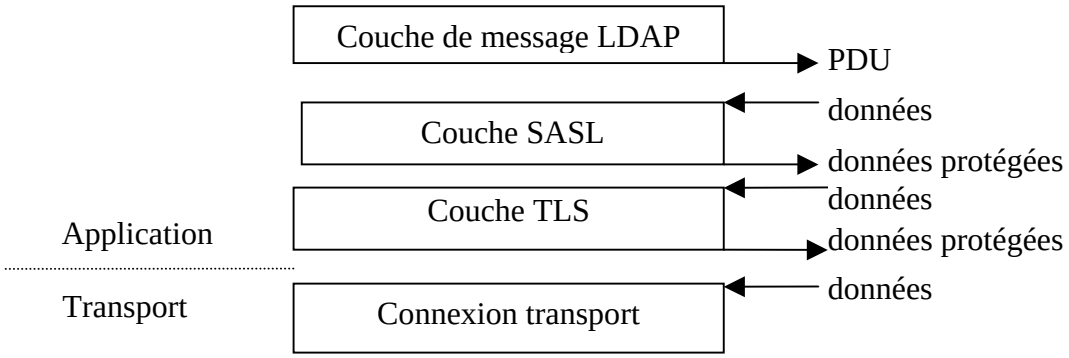
5 Codage de protocole, connexion, et transfert

Ce protocole est conçu pour fonctionner sur des transports fiables orientés connexion, où le flux des données est divisé en octets (unités de 8 bits), chaque octet et chaque bit étant significatif.

Un service sous-jacent, LDAP sur TCP, est défini au paragraphe 5.2. Ces services sont généralement applicables aux applications qui fournissent ou consomment des services de répertoire fondés sur X.500 sur l'Internet. La présente spécification a généralement été écrite en pensant à la transposition TCP. Des spécifications précisant d'autres transpositions pourraient rencontrer divers obstacles.

Les mises en œuvre de LDAP sur TCP DOIVENT mettre en œuvre la transposition décrite au paragraphe 5.2.

Cette figure illustre les relations entre les différentes couches impliquées dans l'échange entre deux homologues de protocole.



5.1 Codage de protocole

Les éléments de protocole de LDAP DOIVENT être codés pour les échanges qui utilisent les règles de codage de base *(Basic Encoding Rules)* [BER] de [ASN.1] avec les restrictions suivantes :

- Seule la forme définie de codage de longueur est utilisée.
- Les valeurs de OCTET STRING sont codées seulement dans la forme primitive.
- Si la valeur d'un type BOOLEAN est vraie, le codage de la valeur de l'octet est réglée à "FF" en hexadécimal.
- Si une valeur d'un type est sa valeur par défaut, il est absent. Seuls certains types BOOLEAN et INTEGER ont des valeurs par défaut dans cette définition de protocole.

Ces restrictions sont destinées à faciliter le surdébit de codage et décodage de certains éléments en BER.

Ces restrictions ne s'appliquent pas aux types ASN.1 encapsulés à l'intérieur des valeurs de OCTET STRING, tels que les valeurs d'attribut, sauf mention contraire.

5.2 Protocole de commande de transmission (TCP)

Les PDU de LDAPMessage codés sont transposés directement en flux d'octets TCP [RFC793] en utilisant le codage fondé sur BER décrit au paragraphe 5.1. Il est recommandé que les mises en œuvre de serveur fonctionnant sur TCP fournissent un surveillant de protocole sur le port LDAP 389 alloué par l'autorité d'allocation des numéros de l'Internet (IANA) [PortReg]. Les serveurs peuvent à la place fournir un surveillant sur un numéro de port différent. Les clients DOIVENT prendre en charge le contact des serveurs sur tout port TCP valide.

5.3 Terminaison de la session LDAP

La terminaison de la session LDAP est normalement initialisée par l'envoi de UnbindRequest par le client (paragraphe 4.3), ou par l'envoi par le serveur d'un avis de déconnexion (paragraphe 4.4.1). Dans ces deux cas, chaque homologue de protocole termine de bon gré la session LDAP en cessant les échanges à la couche de message LDAP, supprimant toute couche SASL, supprimant toute couche TLS, et fermant la connexion de transport.

Un homologue de protocole peut déterminer que la continuation de toute communication serait pernicieuse, et dans ce cas, il peut terminer brusquement la session en cessant la communication et en fermant la connexion de transport.

Dans les deux cas, lorsque la session LDAP est terminée, les opérations non achevées sont traitées comme spécifié au paragraphe 3.1.

6 Considérations sur la sécurité

La présente version du protocole procure des facilités pour une authentification simple en utilisant un mot de passe en clair, ainsi que tout mécanisme SASL [RFC4422]. Installer SASL et/ou des couches TLS peut fournir des services d'intégrité et d'autres services de sécurité des données.

Il est aussi permis que le serveur puisse retourner ses accréditifs au client, s'il choisit de faire ainsi.

L'utilisation d'un mot de passe en clair est fortement déconseillée lorsque le service de transport sous-jacent ne peut pas garantir la confidentialité et peut résulter en la divulgation du mot de passe à des parties non autorisées.

Les serveurs sont encouragés à empêcher les modifications de répertoire par les clients qui se sont authentifiés de façon anonyme [RFC4513].

Les considérations de sécurité pour les méthodes d'authentification, les mécanismes SASL, et TLS sont décrites dans la [RFC4513].

Noter que les échanges d'authentification SASL ne procurent pas la protection de la confidentialité ou de l'intégrité des données pour les champs version ou nom de BindRequest ou du resultCode, diagnosticMessage, ou des champs de renvoi de référence de la BindResponse, ni pour aucune information contenue dans les contrôles attachés aux demandes ou réponses Bind. Et donc, les informations contenues dans ces champs NE DEVRAIENT PAS être tenues pour sûres si elles ne sont pas autrement protégées (comme en établissant des protections à la couche transport).

Les développeurs devraient noter que les divers facteurs de sécurité (incluant les informations d'authentification et d'autorisation et les services de sécurité des données) peuvent changer durant le cours d'une session LDAP ou même durant l'accomplissement d'une opération particulière. Par exemple, des accréditifs peuvent arriver à expiration, des identités d'autorisation ou des contrôles d'accès peuvent changer, ou la ou les couches de sécurité sous-jacentes peuvent être remplacées ou clôturées. Les mises en oeuvre devraient être robustes dans le traitement des changements des facteurs de sécurité.

Dans certains cas, il peut être approprié de continuer l'opération même en présence de changement des facteurs de sécurité. Par exemple, il peut être approprié de continuer une opération Abandon sans considération d'un changement, ou de continuer une opération lorsque le changement a mis à niveau (ou a maintenu) les facteurs de sécurité. Dans d'autres cas, il peut être approprier de faire échouer ou d'altérer la poursuite de l'opération. Par exemple, si des protections de la confidentialité étaient retirées, il pourrait être approprié soit de faire échouer une demande de retourner des données sensibles ou, au minimum, d'exclure de retourner des données sensibles.

Les mises en œuvre qui mettent en mémoire cache des attributs et entrées obtenues via LDAP DOIVENT s'assurer que les contrôles d'accès sont maintenus si ces informations sont à fournir à plusieurs clients, car les serveurs peuvent avoir des politiques de contrôle d'accès qui empêchent de retourner des entrées ou attributs dans les résultats de Search excepté pour des clients authentifiés particuliers. Par exemple, les mémoires cache pourraient ne servir les informations résultantes qu'au client dont la demande a provoqué leur mise en mémoire cache.

Les serveurs peuvent retourner des renvois de références ou des références de résultat de Search qui redirigent les clients sur des serveurs homologues. Il est possible pour une application pirate d'injecter de tels renvois de références dans le flux des données pour essayer de rediriger un client sur un serveur piraté. Les clients sont avertis de se méfier de cela et d'éventuellement rejeter les renvois de références lorsque les mesures de confidentialité ne sont pas en place. Il est conseillé aux clients de rejeter le renvois de référence à partir de l'opération StartTLS.

Les champs matchedDN et diagnosticMessage, ainsi que certaines valeurs de resultCode (par exemple, attributeOrValueExists et entryAlreadyExists), pourraient révéler la présence ou l'absence de données spécifiques dans le répertoire qui est le sujet de l'accès et d'autres contrôles administratifs. Les mises en œuvre de serveur devraient restreindre l'accès pour protéger les informations également dans les conditions normales et les conditions d'erreur.

Les homologues de protocole DOIVENT être prêts à traiter des codages invalides et de longueur arbitraire. Les codages de protocole invalides comprennent : les exceptions de codage BER, les exceptions de chaîne de format et d'UTF-8, les exceptions de débordement, les exceptions de valeurs entières, et les exceptions de fanion marche/arrêt en mode binaire. La suite d'essais LDAPv3 PROTOS [PROTOS-LDAP] fournit d'excellents exemples de ces exceptions et des cas d'essai utilisés pour découvrir les fautes.

Dans le cas où un homologue de protocole perçoit une attaque qui par sa nature pourrait causer des dommages dus à des communications ultérieures à toute couche de la session LDAP, l'homologue de protocole devrait terminer brusquement la session LDAP comme décrit au paragraphe 5.3.

7 Remerciements

Le présent document se fonde sur la RFC 2251 par Mark Wahl, Tim Howes, et Steve Kille. La RFC 2251 a été produite par le groupe de travail ASID de l'IETF.

Il est aussi fondé sur la RFC 2830 par Jeff Hodges, RL "Bob" Morgan, et Mark Wahl. La RFC 2830 a été produite par le groupe de travail LDAPEXT de l'IETF.

Il est aussi fondé sur la RFC 3771 par Roger Harrison et Kurt Zeilenga. La RFC 3771 était une proposition individuelle à l'IETF.

Le présent document a été produit par le groupe de travail LDAPBIS de l'IETF. Parmi les contributeurs significatifs à la révision technique et au contenu figurent Kurt Zeilenga, Steven Legg, et Hallvard Furuseth.

8 Références normatives

[ASN.1] Recommandation UIT-T X.680 (07/2002) | ISO/CEI 8824-1:2002 "Technologies de l'information - Notation de syntaxe abstraite n°1 (ASN.1) : Spécification de la notation de base".

[BER] Recommandation UIT-T X.690 (07/2002) | ISO/CEI 8825-1:2002, "Technologies de l'information - Règles de codage de l'ASN.1 : Spécification des règles de codage de base (BER), Règles de codage canoniques (CER) et règles de codage distinctives (DER)", 2002.

[ISO10646] Universal Multiple-Octet Coded Character Set (UCS) - Architecture et Basic Multilingual Plane, ISO/IEC 10646-1 : 1993.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, septembre 1981.

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, septembre 1981.

[RFC2119] Bradner, S., "Mots clé à utiliser dans les RFC pour indiquer les niveaux d'exigences", BCP 14, RFC 2119, mars 1997.

[RFC3454] Hoffman P. et M. Blanchet, "Preparation of Internationalized Strings ('stringprep')", RFC 3454, décembre 2002.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, novembre 2003.

[RFC3986] Berners-Lee, T., Fielding, R., et L. Masinter, "Identifiant de ressource universel (URI) : Syntaxe générique", STD 66, RFC 3986, janvier 2005.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names et Passwords", RFC 4013, février 2005.

[RFC4234] Crocker, D. et P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, octobre 2005.

[RFC4346] Dierks, T. et E. Rescorla, "The TLS Protocol Version 1.1", RFC 4346, mars 2006.

[RFC4422] Melnikov, A., Ed. et K. Zeilenga, Ed., "Authentification simple et couche de sécurité (SASL)", RFC 4422, juin 2006.

[RFC4510] Zeilenga, K., Ed., "Protocole léger d'accès à un répertoire (LDAP): Feuille de route de la spécification technique", RFC 4510, juin 2006.

[RFC4512] Zeilenga, K., Protocole léger d'accès à un répertoire (LDAP) : Modèles d'informations de répertoire", RFC 4512, juin 2006.

[RFC4513] Harrison, R., Ed., "Protocole léger d'accès à un répertoire (LDAP) : Méthodes d'authentification et mécanismes de sécurité", RFC 4513, juin 2006.

[RFC4514] Zeilenga, K., Ed., "Protocole léger d'accès à un répertoire (LDAP) : Représentation de chaîne des noms distinctifs", RFC 4514, juin 2006.

[RFC4516] Smith, M., Ed. et T. Howes, "Protocole léger d'accès à un répertoire (LDAP) : Localisateurs de ressources uniformes", RFC 4516, juin 2006.

[RFC4517] Legg, S., Ed., "Protocole léger d'accès à un répertoire (LDAP) : Syntaxes et règles de correspondance", RFC 4517, juin 2006.

[RFC4520] Zeilenga, K., "Autorité d'allocation des numéros de l'Internet (IANA) Considérations pour le Protocole léger d'accès à un répertoire (LDAP)", BCP 64, RFC 4520, juin 2006.

[Unicode] The Unicode Consortium, "The Unicode Standard, Version 3.2.0" est défini par "The Unicode Standard, Version 3.0" (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5), amendé par le "Unicode Standard Annex #27: Unicode 3.1" (http://www.unicode.org/reports/tr27/) et par le "Unicode Standard Annex #28: Unicode 3.2" (http://www.unicode.org/reports/tr28/).

[X.500] Recommandation UIT-T X.500, "L'annuaire : aperçu général des concepts, modèles et services", 1993.

[X.511] Recommandation UIT-T X.511, "L'annuaire : définition du service abstrait", 1993.

9 Références informatives

[CharModel] Whistler, K. et M. Davis, "Unicode Technical Report #17, Character Encoding Model", UTR17, <http://www.unicode.org/unicode/reports/tr17/>, August 2000.

[Glossary] The Unicode Consortium, "Unicode Glossary", <http://www.unicode.org/glossary/>.

[PortReg] IANA, "Port Numbers", <http://www.iana.org/assignments/port-numbers>.

[PROTOS-LDAP] University of Oulu, "PROTOS Test-Suite: c06-ldapv3" <http://www.ee.oulu.fi/research/ouspg/protos/testing/ c06/ldapv3/>.

10 Considérations relatives à l'IANA

L'autorité d'allocation des numéros de l'Internet (IANA) a mis à jour le registre des codes de résultat de LDAP pour indiquer que le présent document fournit la spécification technique définitive pour les codes de résultat 0-36, 48-54, 64-70, 80-90. Il est aussi noté qu'une valeur de resultCode (strongAuthRequired) a été renommée (en strongerAuthRequired).

L'IANA a ausi mis à jour le registre des mécanismes du protocole LDAP pour indiquer que le présent document et la [RFC4513] fournissent la spécification technique définitive pour l'opération Extended StartTLS (1.3.6.1.4.1.1466.20037).

IANA a alloué l'identifiant d'objet LDAP 18 [RFC4520] pour identifier le module ASN.1 défini dans le présent document.

Sujet : Demande d'enregistrement d'un identifiant d'objet LDAP
Adresse personnelle & mél à contacter pour complément d'informations :
Jim Sermersheim <jimse@novell.com>
Specification : RFC 4511
Author/Change Controller : IESG
Commentaires : Identifie le module ASN.1 LDAP

Appendice A Codes de résultat LDAP

Cet Appendice normatif précise des considérations supplémentaires concernant les codes de résultat LDAP et donne une brève description générale de chacun des codes de résultat LDAP énumérés au paragraphe 4.1.9.

Des codes de résultat supplémentaires PEUVENT être définis pour être utilisés avec des extensions [RFC4520]. Les mises en œuvre de client DOIVENT traiter tout code de résultat qu'elles ne reconnaissent pas comme une condition d'erreur inconnue.

Les descriptions fournies ici ne tiennent pas entièrement compte des substitutions de code de résultat utilisées pour empêcher la divulgation non autorisée (comme la substitution de noSuchObject pour insufficientAccessRights, ou invalidCredentials pour insufficientAccessRights).

A.1 Codes de résultat de non-erreur

Ces codes de résultat (appelés code de résultats de "non-erreur") n'indiquent pas une condition d'erreur :

success (0),
compareFalse (5),
compareTrue (6),
referral (10), et
saslBindInProgress (14).

Les codes de résultat de succès, compareTrue, et compareFalse, indiquent l'achèvement réussi (et donc, sont appelés des codes de résultat de "succès").

Les codes de résultat referral et saslBindInProgress indiquent le client a besoin d'entreprendre des actions supplémentaires pour achever l'opération.

A.2 Codes de résultat

Les codes de résultats LDAP existants sont décrits comme suit :

success (0)
Indique l'achèvement réussi d'une opération. Note : ce code n'est pas utilisé avec l'opération Compare. Voir compareFalse (5) et compareTrue (6).

operationsError (1)
Indique que l'opération n'est pas dans une séquence appropriée avec les autres opérations (de même type ou de type différent). Par exemple, ce code est retourné si le client essaye StartTLS [RFC4346] alors qu'il y a une autre opération non achevée or si une couche TLS est déjà installée.

protocolError (2)
Indique que le serveur a reçu des données mal formées. Pour la seule opération Bind, ce code est aussi utilisé pour indiquer que le serveur ne prend pas en charge la version de protocole demandée. Pour les seules opérations Extended, ce code est aussi utilisé pour indiquer que le serveur ne prend pas en charge (par conception ou par configuration) l'opération Extended associée au requestName. Pour les opérations de demande qui spécifient plusieurs contrôles, il

peut être utilisé pour indique que le serveur ne peut pas ignorer l'ordre des contrôles spécifié, ou que la combinaison des contrôles spécifiés est invalide ou non spécifié.

timeLimitExceeded (3)
Indique que la limite de temps spécifiée par le client a été dépassée avant l'achèvement de l'opération.

sizeLimitExceeded (4)
Indique que la limite de taille spécifiée par le client a été dépassée avant l'achèvement de l'opération.

compareFalse (5)
Indique que l'opération Compare a été terminée avec succès et que l'assertion a été évaluée à FALSE ou Undefined.

compareTrue (6)
Indique que l'opération Compare a été terminée avec succès et que l'assertion a été évaluée à TRUE.

authMethodNotSupported (7)
Indique que la méthode ou le mécanisme d'authentification n'est pas pris en charge.

strongerAuthRequired (8)
Indique que le serveur exige une (plus) forte authentification afin de terminer l'opération.
Lorsqu'il est utilisé avec l'opération Avis de déconnexion, ce code indique que le serveur a détecté la défaillance inattendue ou la compromission d'une association de sécurité établie entre le client et le serveur.

referral (10)
Indique qu'un renvoi de références doit être poursuivi pour achever l'opération (voir au paragraphe 4.1.10).

adminLimitExceeded (11)
Indique qu'une limite administrative a été dépassée.

unavailableCriticalExtension (12)
Indique qu'un contrôle critique n'est pas reconnu (voir au paragraphe 4.1.11).

confidentialityRequired (13)
Indique que des protections de la confidentialité des données sont exigées.

saslBindInProgress (14)
Indique que le serveur exige que le client envoie une nouvelle demande de lien, avec le même mécanisme SASL, pour continuer le processus d'authentification (voir au paragraphe 4.2).

noSuchAttribute (16)
Indique que l'entrée nommée ne contient pas l'attribut ou la valeur d'attribut spécifié.

undefinedAttributeType (17)
Indique qu'un champ de demande contient une description d'attribut non reconnue.

inappropriateMatching (18)
Indique qu'on a essayé d'utiliser (par exemple, dans une assertion) une règle de correspondance non définie pour le type d'attribut concerné.

constraintViolation (19)
Indique que le client a fourni une valeur d'attribut qui ne se conforme pas aux contraintes que lui impose le modèle des données. Par exemple, ce code est retourné lorsque plusieurs valeurs sont fournies à un attribut qui a une contrainte SINGLE-VALUE.

attributeOrValueExists (20)
Indique que le client a fourni un attribut ou une valeur à ajouter à une entrée, mais l'attribut ou la valeur existe déjà.

invalidAttributeSyntax (21)
Indique qu'une valeur prétendue d'attribut ne se conforme pas à la syntaxe de l'attribut.

noSuchObject (32)
Indique que l'objet n'existe pas dans le DIT.

aliasProblem (33)
Indique qu'un problème d'alias est survenu. Par exemple, le code utilisé pour indiquer qu'un alias a été déréférencé ne nomme pas d'objet.

invalidDNSyntax (34)
Indique qu'un champ LDAPDN ou RelativeLDAPDN (par exemple, base de recherche, entrée cible, ModifyDN newrdn, etc.) d'une demande ne se conforme pas à la syntaxe requise ou contient des valeurs d'attribut qui ne se conforment pas à la syntaxe du type d'attribut.

aliasDereferencingProblem (36)
Indique qu'un problème est survenu en déréférençant un alias. Par exemple, un alias a été rencontré dans une situation où il n'est pas admis ou où l'accès lui est refusé.

inappropriateAuthentication (48)
Indique le serveur exige que le client qui a essayé de se lier en anonyme ou sans fournir d'accréditifs fournisse une forme d'accréditation.

invalidCredentials (49)
Indique que les accréditifs fournis (par exemple, le nom d'utilisateur et le mot de passe) sont invalides.

insufficientAccessRights (50)
Indique que le client n'a pas les droits d'accès suffisants pour effectuer l'opération.

busy (51)
Indique que le serveur est trop occupé pour servir l'opération.

unavailable (52)
Indique que le serveur est en train de fermer ou qu'un sous-système nécessaire pour achever l'opération est débranché.

unwillingToPerform (53)
Indique que le serveur ne veut pas effectuer l'opération.

loopDetect (54)
Indique que le serveur a détecté une boucle interne (par exemple, en déréférençant des alias ou en enchaînant une opération).

namingViolation (64)
Indique que le nom de l'entrée viole des restrictions de dénomination.

objectClassViolation (65)
Indique que l'entrée viole des restrictions de classe d'objet.

notAllowedOnNonLeaf (66)
Indique que l'opération agit à tort sur une entrée ramifiée.

notAllowedOnRDN (67)
Indique que l'opération tente à tort de retirer une valeur qui forme le nom distinctif relatif de l'entrée.

entryAlreadyExists (68)
Indique que la demande ne peut être satisfaite (ajoutée, déplacée ou renommée) car l'entrée cible existe déjà.

objectClassModsProhibited (69)
Indique qu'il est interdit d'essayer de modifier la ou les classes d'objet de l'attribut 'objectClass' d'une entrée.
Par exemple, ce code est retourné lorsque un client tente de modifier la classe d'objet structurale d'une entrée.

affectsMultipleDSAs (71) : Indique que l'opération ne peut être effectuée car elle affecterait plusieurs serveurs (DSA).

other (80) : Indique que le serveur a rencontré une erreur interne.

Appendice B. Définition ASN.1 complète

Le présent Appendice est normatif.

```
Lightweight-Directory-Access-Protocol-V3 {1 3 6 1 1 18}
-- Copyright (C) The Internet Society (2006).  Cette version de ce module ASN.1 fait partie de la RFC 4511 ; voir à la
RFC elle-même les notices légales complètes.
      DEFINITIONS
      IMPLICIT TAGS
      EXTENSIBILITY IMPLIED ::=

      BEGIN

      LDAPMessage ::= SEQUENCE {
          messageID       MessageID,
          protocolOp      CHOICE {
              bindRequest           BindRequest,
              bindResponse          BindResponse,
              unbindRequest         UnbindRequest,
              searchRequest         SearchRequest,
              searchResEntry        SearchResultEntry,
              searchResDone         SearchResultDone,
              searchResRef          SearchResultReference,
              modifyRequest         ModifyRequest,
              modifyResponse        ModifyResponse,
              addRequest            AddRequest,
              addResponse           AddResponse,
              delRequest            DelRequest,
              delResponse           DelResponse,
              modDNRequest          ModifyDNRequest,
              modDNResponse         ModifyDNResponse,
              compareRequest        CompareRequest,
              compareResponse       CompareResponse,
              abandonRequest        AbandonRequest,
              extendedReq           ExtendedRequest,
              extendedResp          ExtendedResponse,
              ...,
              intermediateResponse  IntermediateResponse },
          controls       [0] Controls OPTIONAL }

      MessageID ::= INTEGER (0 ..  maxInt)

      maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --

      LDAPString ::= OCTET STRING – Codé en UTF-8 , caractères [ISO10646]

      LDAPOID ::= OCTET STRING -- Restreint à <numericoid> [RFC4512]

      LDAPDN ::= LDAPString -- Restreint à <distinguishedName> [RFC4514]

      RelativeLDAPDN ::= LDAPString -- Restreint à <name-component> [RFC4514]

      AttributeDescription ::= LDAPString -- Restreint à <attributedescription> [RFC4512]
```

```
AttributeValue ::= OCTET STRING

AttributeValueAssertion ::= SEQUENCE {
    attributeDesc   AttributeDescription,
    assertionValue  AssertionValue }

AssertionValue ::= OCTET STRING

PartialAttribute ::= SEQUENCE {
    type       AttributeDescription,
    vals       SET OF value AttributeValue }

Attribute ::= PartialAttribute(WITH COMPONENTS {
    ...,
    vals (SIZE(1..MAX))})

MatchingRuleId ::= LDAPString

LDAPResult ::= SEQUENCE {
    resultCode         ENUMERATED {
        success                      (0),
        operationsError              (1),
        protocolError                (2),
        timeLimitExceeded            (3),
        sizeLimitExceeded            (4),
        compareFalse                 (5),
        compareTrue                  (6),
        authMethodNotSupported       (7),
        strongerAuthRequired         (8),
            -- 9 réservé --
        referral                     (10),
        adminLimitExceeded           (11),
        unavailableCriticalExtension (12),
        confidentialityRequired      (13),
        saslBindInProgress           (14),
        noSuchAttribute              (16),
        undefinedAttributeType       (17),
        inappropriateMatching        (18),
        constraintViolation          (19),
        attributeOrValueExists       (20),
        invalidAttributeSyntax       (21),
            -- 22-31 non utilisés --
        noSuchObject                 (32),
        aliasProblem                 (33),
        invalidDNSyntax              (34),
            -- 35 reserved for undefined isLeaf --
        aliasDereferencingProblem    (36),
            -- 37-47 non utilisés --
        inappropriateAuthentication  (48),
        invalidCredentials           (49),
        insufficientAccessRights     (50),
        busy                         (51),
        unavailable                  (52),
        unwillingToPerform           (53),
        loopDetect                   (54),
            -- 55-63 non utilisés --
        namingViolation              (64),
        objectClassViolation         (65),
```

```
        notAllowedOnNonLeaf          (66),
        notAllowedOnRDN              (67),
        entryAlreadyExists           (68),
        objectClassModsProhibited    (69),
           -- 70 réservé pour CLDAP --
        affectsMultipleDSAs          (71),
           -- 72-79 non utilisés --
        other                        (80),
        ... },
    matchedDN          LDAPDN,
    diagnosticMessage  LDAPString,
    referral           [3] Referral OPTIONAL }

Referral ::= SEQUENCE SIZE (1..MAX) OF uri URI

URI ::= LDAPString     -- limité aux caractères permis dans les URI

Controls ::= SEQUENCE OF control Control

Control ::= SEQUENCE {
    controlType             LDAPOID,
    criticality             BOOLEAN DEFAULT FALSE,
    controlValue            OCTET STRING OPTIONAL }
BindRequest ::= [APPLICATION 0] SEQUENCE {
    version                 INTEGER (1 ..  127),
    name                    LDAPDN,
    authentication          AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
    simple                  [0] OCTET STRING,
                         -- 1 et 2 reserved
    sasl                    [3] SaslCredentials,
    ... }

SaslCredentials ::= SEQUENCE {
    mechanism               LDAPString,
    credentials             OCTET STRING OPTIONAL }

BindResponse ::= [APPLICATION 1] SEQUENCE {
    COMPONENTS OF LDAPResult,
    serverSaslCreds    [7] OCTET STRING OPTIONAL }

UnbindRequest ::= [APPLICATION 2] NULL

SearchRequest ::= [APPLICATION 3] SEQUENCE {
    baseObject      LDAPDN,
    scope           ENUMERATED {
        baseObject              (0),
        singleLevel             (1),
        wholeSubtree            (2),
        ... },
    derefAliases    ENUMERATED {
        neverDerefAliases       (0),
        derefInSearching        (1),
        derefFindingBaseObj     (2),
        derefAlways             (3) },
    sizeLimit       INTEGER (0 ..  maxInt),
    timeLimit       INTEGER (0 ..  maxInt),
    typesOnly       BOOLEAN,
```

```
    filter          Filter,
    attributes      AttributeSelection }

AttributeSelection ::= SEQUENCE OF selector LDAPString
            -- La LDAPString est restreinte à <attributeSelector> au paragraphe 4.5.1.8

Filter ::= CHOICE {
    et              [0] SET SIZE (1..MAX) OF filter Filter,
    or              [1] SET SIZE (1..MAX) OF filter Filter,
    not             [2] Filter,
    equalityMatch   [3] AttributeValueAssertion,

    substrings      [4] SubstringFilter,
    greaterOrEqual  [5] AttributeValueAssertion,
    lessOrEqual     [6] AttributeValueAssertion,
    present         [7] AttributeDescription,
    approxMatch     [8] AttributeValueAssertion,
    extensibleMatch [9] MatchingRuleAssertion,
    ... }

SubstringFilter ::= SEQUENCE {
    type            AttributeDescription,
    substrings      SEQUENCE SIZE (1..MAX) OF substring CHOICE {
        initial [0] AssertionValue,  -- peut survenir au plus une fois
        any     [1] AssertionValue,
        final   [2] AssertionValue } -- peut survenir au plus une fois
    }

MatchingRuleAssertion ::= SEQUENCE {
    matchingRule    [1] MatchingRuleId OPTIONAL,
    type            [2] AttributeDescription OPTIONAL,
    matchValue      [3] AssertionValue,
    dnAttributes    [4] BOOLEAN DEFAULT FALSE }

SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
    objectName      LDAPDN,
    attributes      PartialAttributeList }

PartialAttributeList ::= SEQUENCE OF
                partialAttribute PartialAttribute

SearchResultReference ::= [APPLICATION 19] SEQUENCE
                    SIZE (1..MAX) OF uri URI

SearchResultDone ::= [APPLICATION 5] LDAPResult

ModifyRequest ::= [APPLICATION 6] SEQUENCE {
    object          LDAPDN,
    changes         SEQUENCE OF change SEQUENCE {
        operation       ENUMERATED {
            add     (0),
            delete  (1),
            replace (2),
            ... },
        modification    PartialAttribute } }

ModifyResponse ::= [APPLICATION 7] LDAPResult
AddRequest ::= [APPLICATION 8] SEQUENCE {
    entry           LDAPDN,
```

```
    attributes      AttributeList }

AttributeList ::= SEQUENCE OF attribute Attribute

AddResponse ::= [APPLICATION 9] LDAPResult

DelRequest ::= [APPLICATION 10] LDAPDN

DelResponse ::= [APPLICATION 11] LDAPResult

ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
    entry           LDAPDN,
    newrdn          RelativeLDAPDN,
    deleteoldrdn    BOOLEAN,
    newSuperior     [0] LDAPDN OPTIONAL }

ModifyDNResponse ::= [APPLICATION 13] LDAPResult

CompareRequest ::= [APPLICATION 14] SEQUENCE {
    entry           LDAPDN,
    ava             AttributeValueAssertion }

CompareResponse ::= [APPLICATION 15] LDAPResult

AbandonRequest ::= [APPLICATION 16] MessageID

ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
    requestName     [0] LDAPOID,
    requestValue    [1] OCTET STRING OPTIONAL }

ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
    COMPONENTS OF LDAPResult,
    responseName    [10] LDAPOID OPTIONAL,
    responseValue   [11] OCTET STRING OPTIONAL }

IntermediateResponse ::= [APPLICATION 25] SEQUENCE {
    responseName    [0] LDAPOID OPTIONAL,
    responseValue   [1] OCTET STRING OPTIONAL }

END
```

Appendice C. Modifications

Cet appendice n'est pas normatif.

Le présent appendice résume les changements de substance apportés aux RFC 2251, RFC 2830, et RFC 3771.

C.1 Modifications à la RFC 2251

La présente section résume les changements de substance apportés aux sections 1, 2, 3.1, et 4, et au reste de la RFC 2251. Le lecteur devrait consulter la [RFC4512] et la [RFC4513] pour le résumé des changements aux autres sections.

C.1.1 Section 1 (Statut de ce mémo)

- Retrait de la note de l'IESG. Dans la post publication de la RFC 2251, les mécanismes d'authentification obligatoires de LDAP ont été normalisés ce qui suffit pour supprimer cette note. Voir les mécanismes d'authentification à la [RFC4513].

C.1.2 Paragraphe 3.1 (Modèle du protocole) et autres

- Retrait des notes donnant l'historique entre LDAP v1, v2, et v3. Ajout à la place d'explications suffisantes pour que le présent document soit autonome.

C.1.3 Paragraphe 4 (Eléments de protocole)

- Précisé où les caractéristiques d'extensibilité de l'ASN.1 s'appliquent au protocole. Ce changement affecte divers types ASN.1 par l'inclusion d'ellipses (...) à certains éléments.
- Supprimée l'exigence que les serveurs qui mettent en œuvre la version 3 ou une version plus récente DOIVENT fournir l'attibut 'supportedLDAPVersion'. Cette exigence ne donnait aucun avantage d'interopérabilité.

C.1.4 Paragraphe 4.1.1 (Enveloppe)

- Il y avait une obligation pour le serveur de retourner un avis de déconnexion et d'abandonner la connexion de transport lorsqu'une PDU est mal formée d'une certaine façon. La mise à jour dit que le serveur DEVRAIT retourner l'avis de déconnexion, et qu'il DOIT terminer la session LDAP.

C.1.5 Paragraphe 4.1.1.1 (ID de message)

- Il est dit que le messageID des demandes DOIT être différent de zéro car le zéro est réservé à l'avis de déconnexion.
- Spécifie quand il est et n'est pas approprié de retourner un messageID déjà utilisé. La RFC 2251 imposait le comportement synchrone des serveurs par une formulation accidentelle.

C.1.6 Paragraphe 4.1.2 (Types de chaînes)

- Il est dit que le LDAPOID est contraint en <numericoid> d'après la [RFC4512].

C.1.7 Paragraphe 4.1.5.1 (Option binaire) et autres

- Retrait de l'option binaire de la spécification. De nombreux problèmes d'interopérabilité sont associés à cette méthode de codage de type d'attribut alterné. La spécification d'un remplacement convenable est en préparation.

C.1.8 Paragraphe 4.1.8 (Attribut)

- Combinaison des définitions de PartialAttribute et d'Attribute, et définition d'Attribute en termes de PartialAttribute.

C.1.9 Paragraphe 4.1.10 (Message Result)

- "errorMessage" est rebaptisé en "diagnosticMessage" cat il est permis de l'envoyer pour des résultats de non-erreur.
- Certaines phrases ont été passées à l'Appendice A, avec un renvoi pour le lecteur.
- Il est permis à matchedDN d'être présent pour d'autres codes de résultat que ceux de la liste de la RFC 2251.
- Le code "strongAuthRequired" est rebaptisé "strongerAuthRequired" pour préciser que ce code peut souvent être retourné pour indiquer qu'une authentification plus forte est nécessaire pour effectuer une opération donnée.

C.1.10 Paragraphe 4.1.11 (Renvoi de référence)

- Définition des renvois de référence en termes d'URI plutôt que d'URL.
- Retrait de l'exigence que tous les URI de renvoi de référence DOIVENT être également capables de poursuivre l'opération. La déclaration était ambiguë et ne donnait pas d'instructions sur la façon de le faire.
- Ajout de l'exigence que les clients NE DOIVENT PAS boucler entre les serveurs.
- Précisé les instructions d'utilisation des LDAPURL dans les renvois de référence, et ajout d'une recommandation que la partie domaine soit présente.
- Retrait des impératifs qui exigeaient des clients qu'ils utilisent des URL de façon spécifique pour poursuivre une opération. Cela n'apportait rient à l'interopérabilité.

C.1.11 Paragraphe 4.1.12 (Controls)

- Spécifie comment les valeurs de control définie en termes d'ASN.1 sont à coder.
- Noté que le champ criticality n'est appliqué que pour les messages de demande (excepté UnbindRequest), et doit être ignoré lorsque présent sur les messages de réponse et UnbindRequest.
- Spécifié que les contrôles non critiques peuvent être ignorés à la discrétion du serveur. Il y avait une confusion dans la formulation originelle qui conduisait certains à croire que les contrôles reconnus ne peuvent pas être ignorés pour autant qu'ils soient associés à une demande appropriée.
- Ajout d'une phrase concernant les combinaisons de contrôles et l'ordre des contrôles sur un message.
- Spécifié que lorsque la sémantique de la combinaison des contrôles est indéfinie ou inconnue, il en résulte un protocolError.
- Changé "Le serveur DOIT être prêt" en "Les mises en œuvre DOIVENT être prêtes" au huitième alinéa pour refléter que le client et les mises en œuvre de serveur doivent être capables de traiter cela (car tous deux analysent les controls).

C.1.12 Paragraphe 4.2 (Opération Bind)

- Il devient obligatoire que les serveurs retournent protocolError lorsque la version n'est pas prise en charge.
- Précisé le comportement lorsque l'authentification simple est utilisée, le nom est vide, et le mot de passe non vide.
- Exigé que les serveurs ne déréférencent pas les alias pour Bind. Cela a été ajouté pour la cohérence avec les autres opérations et pour aider à assurer la cohérence des données.
- Exigé que les mots de passe textuels soient transférés en Unicode codé en UTF-8, et ajout de recommandations sur la préparation de chaîne. Cela pour aider à assurer l'interopérabilité de mots de passe envoyés de différents clients.

C.1.13 Paragraphe 4.2.1 (Séquençage de la demande Bind)

- Ce paragraphe a été réorganisé pour en améliorer la lisibilité, et du texte a été ajouté pour préciser l'état d'authentification échoué et les opérations Bind abandonnées.
- Retrait de "Si un mécanisme de chiffrement de transfert SASL ou d'intégrité a été négocié, si ce mécanisme ne prend pas en charge le changement des accréditifs d'une identité à une autre, le client DOIT à la place établir alors une nouvelle connexion." S'il y a dépendance entre plusieurs négociations d'un mécanisme SASL particulier, la spécification technique pour ce mécanisme SASL précise comme les applications vont les traiter. LDAP ne dvrait pas exiger de traitement particulier.
- Abandon de l'impératif DOIT au troisième alinéa pour la cohérence avec la [RFC2119].
- Il devient obligatoire que les clients n'envoient pas d'opérations non-Bind alors qu'un Bind est en cours, et il est suggéré que les serveurs ne les traitent pas si ils les reçoivent. Ceci est nécessaire pour assurer un séquencement approprié du Bind en relation avec les autres opérations.

C.1.14 Paragraphe 4.2.3 (Réponse Bind)

- La plus grande partie du texte relatif aux erreurs a été déplacée à l'Appendice A, et du texte concernant certaines erreurs utilisées en conjonction avec l'opération Bind a été ajouté.
- IL devient interdit que le serveur spécifie serverSaslCreds lorsque ce n'est pas approprié.

C.1.15 Paragraphe 4.3 (Opération Unbind)

- IL est spécifié que les deux homologues cessent la transmission et terminent la session LDAP pour l'opération Unbind.

C.1.16 Paragraphe 4.4 (Notification non sollicitée)

- Ajout d'instructions pour des spécifications futures de Notifications non sollicitées.

C.1.17 Paragraphe 4.5.1 (Demande Search)

- Les attributs SearchRequest sont maintenant définis comme un type AttributeSelection plutôt que comme AttributeDescriptionList, et un ABNF est fourni.
- Les attributs SearchRequest peuvent contenir des descriptions d'attribut dupliquées. Cela était interdit précédemment. Maintenant, les serveurs ont pour instruction d'ignorer les noms suivants s'ils sont dupliqués. Cela a été assoupli afin de permettre de demander différents noms abrégés et aussi d'OID pour un attribut.

- Le présent filtre de recherche évalue à Undefined lorsque l'attribut spécifié n'est pas connu du serveur. Il était utilisé pour évaluer à FALSE, ce qui causait un comportement incohérent avec ce qu'attendaient la plupart, particulièrement lorsque l'opérateur 'not' était utilisé.
- Le type de sous-chaînes SubstringFilter de choix du filtre est maintenant défini avec une limite inférieure de 1.
- Les types sous-chaînes SubstringFilter 'initial, 'any', et 'final' sont maintenant AssertionValue plutôt que LDAPString. Ajout aussi d'impératifs déclarant que 'initial' (s'il est présent) doit figurer en premier sur la liste, et 'final' (s'il est présent) doit figurer en dernier.
- Précision de la sémantique des choix derefAliases. Il y avait des questions sur la façon dont derefInSearching s'applique à l'objet de base dans un wholeSubtree Search.
- Ajout d'instructions pour equalityMatch, substrings, greaterOrEqual, lessOrEqual, et approxMatch.

C.1.18 Paragraphe 4.5.2 (Résultat Search)

- IL est recommandé que les serveurs n'utilisent pas les noms abrégés d'attribut lorsqu'ils savent qu'ils sont ambigus ou peuvent causer des problèmes d'interopérabilité.
- Retrait de toute mention de ExtendedResponse due à l'absence de toute mise en œuvre.

C.1.19 Paragraphe 4.5.3 (Références de continuation dans le résultat de Search)

- Changements similaires à ceux faits au paragraphe 4.1.11.

C.1.20 Paragraphe 4.5.3.1 (Exemple)

- Les exemples ont été adaptés aux changements apportés au paragraphe 4.5.3.

C.1.21 Paragraphe 4.6 (Opération Modify)

- Remplacé AttributeTypeAndValues par Attribute qui lui est équivalent.
- Spécifié les changements de types de modification qui pourraient temporairement violer le schéma. Certains lecteurs avaient l'impression que toute violation temporaire de schéma était permise.

C.1.22 Paragraphe 4.7 (Opération Add)

- Alignement de l'opération Add avec X.511 en ce que les attributs du RDN sont utilisés en conjonction avec la liste des attributs pour créer l'entrée. Précédemment, Add exigeait que les valeurs distinctives soient présentes dans la liste des attributs.
- Retrait de l'exigence que l'attribut objectClass soit spécifié car certains types DSE n'exigent pas cet attribut. Une formulation générale a été ajoutée à la place, exigeant que l'entrée ajoutée se conforme au modèle de données.
- Retrait de la recommandation concernant le placement des objets. Ceci est couvert par le document de modèle de données.

C.1.23 Paragraphe 4.9 (Opération Modify DN)

- Il est exigé des serveurs qu'ils ne déréférencent pas les alias pour Modify DN. Ceci a été ajouté pour la cohérence avec les autres opérations et pour aider à assurer la cohérence des données.
- Permet l'échec de Modify DN lors de déplacement entre des contextes de dénomination.
- Spécifié ce qui arrive lorsque les attributs de newrdn ne sont pas présents sur l'entrée.

C.1.24 Paragraphe 4.10 (Opération Compare)

- Spécifié que compareFalse signifie que Compare a eu lieu et que le résultat est faux. Il y avait une confusion qui conduisait les gens à croire qu'une confrontation Undefined résultait en compareFalse.
- Il est exigé des serveurs qu'ils ne déréférencent pas les alias pour Compare. Ceci a été ajouté pour la cohérence avec les autres opérations et pour aider à assurer la cohérence des données.

C.1.25 Paragraphe 4.11 (Opération Abandon)

- Explique que comme Abandon ne retourne pas de réponse, les clients ne devraient pas l'utiliser s'ils ont besoin de connaître le résultat.
- Spécifié que Abandon et Unbind ne peuvent pas être abandonnés.

C.1.26 Paragraphe 4.12 (Opération Extended)

- Spécifié comment coder les valeurs des opérations Extended définies en ASN.1.
- Ajout d'instructions sur ce en quoi consistent les spécifications de l'opération Extended.
- Ajout d'une recommandation que les serveurs avertissent des opérations Extended qu'ils prennent en charge.

C.1.27 Paragraphe 5.2 (Protocoles de transfert)

- Déplacement d'instructions spécifiques du renvoi de références dans les paragraphes qui s'y rapportent.

C.1.28 Paragraphe 7 (Considérations sur la sécurité)

- Reformulation des notes concernant la non protection par SASL de certains aspects des messages LDAP Bind.
- Noté que les serveurs ont encouragés à empêcher les modifications de répertoire par les clients qui ont choisi l'authentification anonyme [RFC4513].
- Ajout d'une note concernant la possibilité de changements de facteurs de sécurité (authentification, autorisation, et confidentialité des données).
- Avertissement sur les dangers de suivre des renvois de références pouvant avoir été injecté dans le flux des données.
- Noté que les serveurs devraient protéger également les informations, qu'on soit ou non en condition d'erreur et mentionné spécifiquement matchedDN, diagnosticMessage, et resultCodes.
- Ajout d'une note concernant les codages mal formés et longs.

C.1.29 Appendice A (Définition ASN.1 complète)

- Ajout de "EXTENSIBILITY IMPLIED" à la définition ASN.1.
- Retrait de AttributeType. Non utilisé.

C.2 Modifications à la RFC 2830

Ce paragraphe résume les changements de substance apportés aux paragraphes de la RFC 2830. Les lecteurs devraient consulter la [RFC4513] pour le résumé des changements aux autres sections.

C.2.1 Paragraphe 2.3 (Réponses autres que "success")

- Retrait de la formulation indiquant que les renvois de références peuvent être retournés à partir de StartTLS.
- Retrait de l'exigence que seul un petit ensemble de codes de résultat peut être retourné. Certains codes de résultat sont nécessaires dans certains scénarios, mais n'importe quel autre peut être retourné si approprié.
- Retrait de l'exigence que ExtendedResponse.responseName soit présent. Il y a des circonstances dans lesquelles c'est impossible, et qui exigent un alignement avec la formulation du paragraphe 4.12.

C.2.1 Paragraphe 4 (Clôture d'une connexion TLS)

- Reformulation de la plus grande partie de ce paragraphe pour l'aligner avec les définitions des couches du protocole LDAP.
- Retrait des instructions sur la clôture brusque qui est couverte dans d'autres parties du document (précisément, au paragraphe 5.3)

C.3 Modifications à la RFC 3771

- Reformulation pour s'harmoniser avec le présent document. En général, la sémantique a été préservée. Les formulations de support et de base considérées comme redondantes du fait de leur présence dans le présent document ont été omises.
- Il est spécifié que les réponses Intermediate à une demande peuvent être de différents types, et un des types de réponse peut être spécifié comme n'ayant pas de valeur de réponse.

Adresse de l'éditeur

Jim Sermersheim
Novell, Inc.
1800 South Novell Place
Provo, Utah 84606, USA
tél 801 861-3088
mél : jimse@novell.com

Déclaration de copyright

Propriété intellectuelle

Remerciement

Le financement de la fonction d'édition des RFC est actuellement fourni par l'activité de soutien administratif de l'IETF (IASA, *Administrative Support Activity*).