Abstract

This document defines support for the Server Sorting Control, the Virtual List Control, the Persistent Search Control, the Proxied Authorization Control, the Authentication Request Control, and the Duplicate Entry Control in the Java LDAP API. Controls are an LDAP protocol version 3 extension to allow passing arbitrary control information along with a standard request to a server, and to receive arbitrary information back with a standard result.
1. Introduction....................................................4
2. Overview of the LDAP Control classes.............................4
3. The java LDAP Control classes......................................6
3.1 public class LDAPVirtualListControl............................6
3.1.1 Constructors..................................................6
3.1.2 getAfterCount..................................................7
3.1.3 getBeforeCount..................................................7
3.1.4 getListSize.....................................................8
3.1.5 setListSize.....................................................8
3.1.6 setRange.........................................................8
3.1.7 getContext......................................................8
3.1.8 setContext......................................................8
3.2 public class LDAPVirtualListResponse..........................9
3.2.1 getContentCount................................................9
3.2.2 getFirstPosition...............................................9
3.2.3 getResultCode..................................................9
3.3 public class LDAPSortControl..................................9
3.3.1 Constructors..................................................9
3.4 public class LDAPSortResponse.................................10
3.4.1 getFailedAttribute............................................10
3.4.2 getResultCode..................................................10
3.5 public class LDAPPersistSearchControl.......................10
3.5.1 Constructors..................................................11
3.5.2 getChangeTypes...............................................11
3.5.3 getReturnControls.............................................11
3.5.4 setChangeTypes...............................................11
3.5.5 setChangesOnly...............................................12
3.5.6 setReturnControls.............................................12
3.6 public class LDAPEntryChangeControl........................12
3.6.1 getChangeNumber...............................................12
3.6.2 getChangeType...............................................12
3.6.3 getPreviousDN...............................................12
3.7 public class LDAPProxiedAuthControl........................12
3.7.1 Constructors..................................................13
3.8 public class LDAPAuthRequestControl........................13
3.8.1 Constructors..................................................13
3.9 public class LDAPAuthResponse.................................13
3.9.1 getAuthorizedIdentity........................................13
3.10 public class LDAPDuplicateEntryControl....................13
3.10.1 Constructors..................................................14
3.11 public class LDAPDuplicateEntryResponse....................14
3.11.1 getResultCode................................................14
3.11.2 getErrorMessage.............................................14
3.11.3 getFailedAttribute..........................................14
4. Java LDAP Control helper classes..................................15
4.1 public class LDAPSortKey........................................15
4.1.1 Constructors..................................................15
4.1.2 getKey.........................................................16
4.1.3 getReverse....................................................16
4.1.4 getMatchRule..................................................16
1. Introduction

Version 3 of the LDAP protocol provides a means of supplying arbitrary additional information along with a request to an LDAP server, and receiving arbitrary additional response information. A few applications of the Control mechanism have been identified as having general interest, and the protocol defined for their transmission [5] and [6]. This document defines how support for the Server Sorting Control, the Virtual List Control, the Persistent Search Control, the Proxied Authorization Control, the Authentication Request Control, and the Duplicate Entry Control are supported in the Java LDAP API. The Java LDAP API in general is described in [2]. The Control protocol extension is described in [1], section 4.1.12, and applications of it in [5] and [6].

2. Overview of the LDAP Control classes

LDAPControl is part of a basic LDAP class package. Specific applications/implementations of Controls are in a subpackage called "controls".

The base class LDAPControl is defined in [2] as:

```java
public class LDAPControl implements Cloneable
```

An LDAPControl encapsulates optional additional parameters or constraints to be applied to LDAP operations. If set as a Server Control, it is sent to the server along with operation requests. If set as a Client Control, it is not sent to the server, but rather interpreted locally by the client. LDAPControl is an LDAPv3 extension, and is not supported in an LDAPv2 environment.

Constructors

```java
public LDAPControl(String id,
                     boolean critical,
                     byte vals[])
```

Parameters are:

- `id` The type of the Control, as a string.
- `critical` True if the LDAP operation should be discarded if the server does not support this Control.
- `vals` Control-specific data.

`getID`

```java
public String getID()
```
Returns the identifier of the control.

isCritical

public boolean isCritical()

Returns true if the control must be supported for an associated operation to be executed.

ggetValue

public byte[] getValue()

Returns the control-specific data of the object.

The following Controls are defined for the controls subpackage:

LDAPVirtualListControl

Encapsulates requests for a subset of a virtual list of search results.

LDAPVirtualListResponse

Encapsulates the response of a server to a virtual list request.

LDAPSortControl

Encapsulates a requested sorting order for search results returned by a server.

LDAPSortResponse

Encapsulates a server’s response to a request that included a sort control.

LDAPPersistSearchControl

Used to start a persistent search, one which runs continuously, returning results as the Directory is modified.

LDAPEntryChangeControl

Returned by the server for changed entries during a persistent search.

LDAPProxiedAuthControl

Used to request that an operation be executed as an identity specified in the control.

LDAPAuthResponse

May be returned by a server in a bind response to indicate the authorized identity of a successful bind.

LDAPDuplicateEntryControl

Requests that the server return separate entries for each value held in a specified attribute.
LDAPDuplicateEntryResponse   Returned by the server on completion of a search request that included a duplicate entry control.

The following helper class is used by LDAPSortControl:

LDAPSsortKey               Defines an attribute to sort by, the sorting order, and optionally a matching rule to use in sorting.

3. The java LDAP Control classes

3.1 public class LDAPVirtualListControl
    extends LDAPControl

LDAPVirtualListControl is a Server Control to specify that results from a search are to be returned in pages, subsets of the entire virtual result set. On success, an updated LDAPVirtualList object is returned as a response Control, containing information on the virtual list size and the actual first index. This object can then be updated by the client with a new requested position or length and sent to the server to obtain a different segment of the virtual list. The protocol elements are defined in [6].

3.1.1 Constructors

public LDAPVirtualListControl( String jumpTo,
                               int beforeCount,
                               int afterCount )

public LDAPVirtualListControl( String jumpTo,
                               int beforeCount,
                               int afterCount,
                               String context )

Constructs a virtual list control using the specified filter expression for the first entry, which defines the extent of the virtual search results, and the number of entries before and after a located index to be returned.

public LDAPVirtualListControl( int startIndex,
                               int beforeCount,
                               int afterCount,
                               int contentCount )

public LDAPVirtualListControl( int startIndex,
                               int beforeCount,
                               int afterCount,
int contentCount,
     String context)

Use this constructor when the size of the virtual list is known, to
fetch a subset.

Parameters are:

jumpTo   A search expression that defines the first
element to be returned in the virtual search
results. The filter expression in the search
operation itself may be, for example,
"objectclass=person" and the jumpTo expression in
the virtual list control may be "cn=m*", to
retrieve a subset of entries starting at or
centered around those with a common name
beginning with the letter "M".

beforeCount The number of entries before startIndex (the
reference entry) to be returned.

afterCount  The number of entries after startIndex to be
returned.

startIndex  The index of the reference entry to be returned.

contentCount The total number of entries assumed to be in the
list. This is a number returned on a previous
search, in the LDAPVirtualListResponse. The
server may use this number to adjust the returned
subset offset.

context     Used by some implementations to process requests
more efficiently. The context should be null on
the first search, and thereafter it should be
whatever was returned by the server in the
virtual list response control.

3.1.2 getAfterCount

public int getAfterCount()

Returns the number of entries after the top/center one to return per
page of results.

3.1.3 getBeforeCount

public int getBeforeCount()

Returns the number of entries before the top/center one to return per
3.1.4 getListSize

public int getListSize()

Returns the size of the virtual search results list. For a newly constructed control – one which is not the result of parseResponse on a control returned by a server – the method returns -1.

3.1.5 setListSize

public void setListSize( int size )

Sets the assumed size of the virtual search results list. This will typically be a number returned on a previous virtual list request in an LDAPVirtualListResponse.

3.1.6 setRange

public void setRange( int listIndex,
                    int beforeCount,
                    int afterCount )

Sets the center or starting list index to return, and the number of results before and after.

Parameters are:

- listIndex: The center or starting list index to be returned.
- beforeCount: The number of entries before "listIndex" to be returned.
- afterCount: The number of entries after "listIndex" to be returned.

3.1.7 getContext

public String getContext()

Returns the cookie used by some servers to optimize the processing of virtual list requests.

3.1.8 setContext

public void setContext( String context )
Sets the cookie used by some servers to optimize the processing of virtual list requests. It should be the context field returned in a virtual list response control for the same search.

3.2 public class LDAPVirtualListResponse
   extends LDAPControl

   LDAPVirtualListResponse is a Server Control returned by the server in response to a virtual list search request.

3.2.1 getContentCount

   public int getContentCount ()

   Returns the size of the virtual search results list

3.2.2 getFirstPosition

   public int getFirstPosition ()

   Returns the index of the first entry returned

3.2.3 getResultCode

   public int getResultCode ()

   Returns the result code for the virtual list request

3.2.4 getContext

   public String getContext()

   Returns the cookie used by some servers to optimize the processing of virtual list requests.

3.3 public class LDAPSortControl
   extends LDAPControl

   LDAPSortControl is a Server Control to specify how search results are to be sorted by the server (see [5]). If a server does not support sorting in general or for a particular query, the results will be returned unsorted, along with a control indicating why they were not sorted (or that sort controls are not supported). If the control was marked "critical", the whole search operation will fail if the sort control is not supported.
3.3.1 Constructors

public LDAPSortControl( LDAPSortKey key, boolean critical)
Constructs a sort control with a single key.

public LDAPSortControl( LDAPSortKey[] keys, boolean critical)
Constructs a sort control with multiple sort keys.

Parameters are:

key            A sort key object, which specifies attribute, order, and optional matching rule. See 4.1
keys           An array of sort key objects, to be processed in order.
critical       True if the search operation is to fail if the server does not support this control.

3.4 public class LDAPSortResponse
    extends LDAPControl

LDAPSortResponse is returned by a server in response to a search request with a sort control.

3.4.1 getFailedAttribute

public String getFailedAttribute()

If not null, this returns the attribute that caused the sort operation to fail.

3.4.2 getResultCode

public int getResultCode ()

Returns the result code from the sort, as defined in [1], section 4.1.10.

3.5 public class LDAPPersistSearchControl
    extends LDAPControl

The LDAPPersistSearchControl class is used to start a persistent search, one that doesn’t end after returning any initial results, but
continues to monitor changes in a designated part of a Directory, reporting the results as changes are made. The protocol elements are defined in [4].

3.5.1 Constructors

public LDAPPersistSearchControl(int changeTypes,
                                boolean changesOnly,
                                boolean returnControls,
                                boolean isCritical)

Parameters are:

changeTypes The change types to be monitored as a logical OR of any or all of these types: ADD, DELETE, MODIFY, and/or MODDN.

changesOnly true if the initial search is to be skipped.

returnControls true if entry change controls are to be returned with the search results.

isCritical true if the search is to be abandoned if the server doesn’t support this control.

3.5.2 getChangeTypes

public int getChangeTypes()

Returns the change types to be monitored as a logical OR of any or all of these types: ADD, DELETE, MODIFY, and/or MODDN.

3.5.3 getReturnControls

public boolean getReturnControls()

Returns true if entry change controls are to be returned with the search results.

3.5.4 setChangeTypes

public void setChangeTypes(int types)

Sets change types to be monitored.

Parameters are:

types The change types to be monitored as a logical OR of any or all of these types: ADD, DELETE, MODIFY, and/or MODDN.
3.5.5 setChangesOnly

public void setChangesOnly(boolean changesOnly)

Requests that only changes be returned - skip the initial search.

Parameters are:

changesOnly    true to skip the initial search.

3.5.6 setReturnControls

public void setReturnControls(boolean returnControls)

Requests that entry change controls are returned with the search results.

Parameters are:

returnControls       true to return entry change controls.

3.6 public class LDAPEntryChangeControl
    extends LDAPControl

An LDAPEntryChangeControl object may be returned by a server when an entry changes, during a persistent search.

3.6.1 getChangeNumber

public int getChangeNumber ()

Returns record number of the change in the server’s change log.

3.6.2 getChangeType

public int getChangeType()

Returns one of these types: ADD, DELETE, MODIFY, and/or MODDN.

3.6.3 getPreviousDN

public String getPreviousDN ()

Returns the previous DN of the entry, if it was renamed.

3.7 public class LDAPProxiedAuthControl
    extends LDAPControl

Expires October 2001
The LDAPProxiedAuthControl class is used to request that the operation it accompanies be executed using an identity specified in the control. The protocol elements are defined in [7].

3.7.1 Constructors

public LDAPProxiedAuthControl ( String dn,  
  boolean isCritical)

Parameters are:

  dn             The identity to execute as.
  isCritical     true if the search is to be abandoned if the server doesn’t support this control.

3.8 public class LDAPAuthRequestControl  
  extends LDAPControl

An LDAPAuthRequestControl object may be included with a bind request to indicate that the server should return an authentication response control with the final bind response if it succeeds. The protocol elements are defined in [8].

3.8.1 Constructors

public LDAPAuthRequestControl ()

3.9 public class LDAPAuthResponse  
  extends LDAPControl

An LDAPAuthResponse object is returned by a server on a successful bind if an authentication request control was provided on the bind request, and if the server supports the controls.

3.9.1 getAuthorizedIdentity

public String getAuthorizedIdentity ()

Returns the identity resulting from the bind, if successful. It returns null for anonymous authentication.

3.10 public class LDAPDuplicateEntryControl  
  extends LDAPControl

LDAPDuplicateEntryControl can be used with an LDAP search by a client to request that the server return separate entries for each value held in the specified attribute. For example if a user has
multiple telephoneNumber values in an entry, each telephoneNumber is
returned as a separate result. The protocol elements are defined in
[9].

3.10.1 Constructors

public LDAPDuplicateEntryControl( String[] attrs,
       boolean partialOK )

Constructs a duplicate entry control using the specified attribute
list.

Parameters are:

   attrs          Specifies an array of attributes.
   partialOK      true if the server is allowed to apply the
                   control to a subset of search results.

3.11 public class LDAPDuplicateEntryResponse
       extends LDAPControl

   LDAPDuplicateEntryResponse is a server control that is returned by
   the LDAP server with the searchResultDone message in response to a
   duplicate entry control request. It includes a resultcode, optional
   error message and optional name of first attribute to be matched.

3.11.1 getResultCode

   public int getResultCode()

   Returns the result code from the sort, as defined in [9], section
   4.2.2.

3.11.2 getErrorMessage

   public String getErrorMessage()

   Returns the error string returned by the LDAP server if an error
   occured when processing this control.

3.11.3 getFailedAttribute

   public String getFailedAttribute()

   If an error occurs, the LDAP server may return the name of the first
   attribute that was included in the duplicate entry control request.
   This method is used to retrieve this attribute name if an error
   result code was returned.
4. Java LDAP Control helper classes

4.1 public class LDAPSortKey

Encapsulates parameters for sorting search results.

4.1.1 Constructors

public LDAPSortKey( String keyDescription )

Constructs a new LDAPSortKey object using a, possibly complex, sorting specification.

public LDAPSortKey( String key, boolean reverse)

Constructs a new LDAPSortKey object using an attribute name and a sort order.

public LDAPSortKey( String key, boolean reverse, String matchRule)

Constructs a new LDAPSortKey object using an attribute name, a sort order, and a matching rule.

Parameters are:

keyDescription A single attribute specification to sort by. If prefixed with "-", reverse order sorting is requested. A matching rule OID may be appended following ":".

Examples:
"cn"
"-cn"
"-cn:1.2.3.4.5"

key An attribute name, e.g. "cn".

reverse True to sort in reverse collation order.

matchRule The object ID (OID) of a matching rule used for collation. If the object will be used to request server-side sorting of search results, it should be the OID of a matching rule known to be supported by that server.
4.1.2 getKey

public String getKey()

Returns the attribute to be used for collation.

4.1.3 getReverse

public boolean getReverse()

Returns true if the sort key specifies reverse-order sorting.

4.1.4 getMatchRule

public String getMatchRule()

Returns the OID to be used as matching rule, or null if none is to be used.

5. Security Considerations

See [2] for security considerations in the java LDAP API.

6. Bibliography


7. Authors’ Addresses

Rob Weltman
Coscend Corp.
3290 West Bayshore Road
Palo Alto, CA 94303
USA
+1 650 461 1708
robw@coscend.com

Javed Khan
Novell, Inc.
1800 South Novell Place
Provo, UT 84606
USA
jkhan@novell.com
8. Appendix A - Sample usage of the java LDAP controls

Doing a search with results sorted on the server

```java
import org.ietf.ldap.*;
import org.ietf.ldap.controls.***;
import java.util.*;

public class SearchJensenSorted {
    public static void main( String[] args ) {
        try {
            LDAPConnection ld = new LDAPConnection();
            /* Connect to server */
            String MY_HOST = "localhost";
            int MY_PORT = 389;
            ld.connect( MY_HOST, MY_PORT );

            /* search for all entries with surname of Jensen */
            String MY_FILTER = "sn=Jensen";
            String MY_SEARCHBASE = "o=Ace Industry, c=US";

            /* Get the common name, uid, and telephone number */
            String[] attrs = new String[3];
            attrs[0] = "cn";
            attrs[1] = "telephonenumber";
            attrs[2] = "uid";

            /* Sort by lastname, firstname */
            LDAPSortKey[] keys = new LDAPSortKey[2];
            keys[0] = new LDAPSortKey( "sn" );
            keys[1] = new LDAPSortKey( "givenname" );
            LDAPSortControl sort = new LDAPSortControl( keys, true );
            LDAPSearchConstraints cons = ld.getSearchConstraints();
            cons.setServerControls( ld.SERVERCONTROLS, sort );

            LDAPSearchResults res =
                ld.search( MY_SEARCHBASE,
                           LDAPConnection.SCOPE_ONE,
                           MY_FILTER,
                           attrs,
                           false,
                           cons );

            /* Loop on results until finished */
            while ( res.hasMoreElements() ) {
                /* Next directory entry */
                LDAPEntry findEntry = (LDAPEntry)res.nextElement();
                System.out.println( findEntry.getDN() );

                /* Get the attributes of the entry */
            }
        }
    }
}
```
LDAPAttributeSet findAttrs = findEntry.getAttributeSet();
Enumeration enumAttrs = findAttrs.getAttributes();
System.out.println( "Attributes: " );

/* Loop on attributes */
while ( enumAttrs.hasMoreElements() ) {
    LDAPAttribute anAttr =
        (LDAPAttribute)enumAttrs.nextElement();
    String attrName = anAttr.getName();
    System.out.println( "" + attrName );
    /* Loop on values for this attribute */
    Enumeration enumVals = anAttr.getStringValues();
    while ( enumVals.hasMoreElements() ) {
        String aVal =
            ( String )enumVals.nextElement();
        System.out.println( "" + aVal );
    }
}

/* Check if the server had something to say about the
sort request */
LDAPControl[] controls = ld.getResponseControls();
if ( controls != null ) {
    for( int i = 0; i < controls.length; i++ ) {
        if ( controls[i] instanceof LDAPSortResponse ) {
            String bad = ((LDAPSortResponse)controls[i]).
                getFailedAttribute();
            int res = ((LDAPSortResponse)controls[i]).
                getResultCode();
            if ( res != 0 ) {
                System.out.println( "Error code: " + res );
                if ( bad != null ) {
                    System.out.println( "Offending " +
                        "attribute: " + bad );
                } else {
                    System.out.println( "No offending " +
                        "attribute " + "returned" );
                }
            }
            break;
        }
    }
} catch( LDAPException e ) {
    System.out.println( e.toString() );
}
/* Done, so disconnect */
if ( ld.isConnected() )
    ld.disconnect();
}
Using virtual list controls - an application using JFC

```java
import org.ietf.ldap.*;
import org.ietf.ldap.controls.*;

// Call this to initialize the list box, whenever the search
// conditions change.
// "filter" may be "objectclass=person", for example
void initListBox( String host, int port,
                 String base, String filter ) {
// Create list box if not already done
if ( _dataList == null ) {
    _dataList = new JList();
    JScrollPane scrollPane = new JScrollPane(_dataList);
    add( scrollPane );
}
// Create a virtual data model
vlistModel model = new vlistModel( host, port, base, filter );
// Keep a buffer of one page before and one after
model.setPageSize( getScrollVisibleSize() );
_dataList.setModel( model );
}

// Data model to supply buffer list data
class vlistModel extends AbstractListModel {
    vlistModel( String host, int port, String base, String filter ) {
        _base = base;
        _filter = filter;
        // Connect to the server
        try {
            _ldc = new LDAPConnection();
            System.out.println( "Connecting to " + host + ":" + port );
            _ldc.connect( host, port );
        } catch ( LDAPException e ) {
            System.out.println( e );
            _ldc = null;
        }
    }
    // Called by JList to get virtual list size
    public int getSize() {
        if ( !_initialized ) {
            _initialized = true;
            _pageControls = new LDAPControl[2];
            // Paged results also require a sort control
            _pageControls[0] =
                new LDAPSortControl( new LDAPSortKey("cn"),
                                 true );
            // Do an initial search to get the virtual list size
            // Keep one page before and one page after the start
        }
```
```java
_beforeCount = _pageSize;
_afterCount = _pageSize;
// Create the initial paged results control
LDAPVirtualListControl cont =
    new LDAPVirtualListControl("A",
        _beforeCount,
        _afterCount);
_pageControls[1] = cont;
_vlc = (LDAPVirtualListControl)_pageControls[1];
getPage(0);
}
return _size;
}
// Get a page starting at first (although we may also fetch
// some preceding entries)
boolean getPage( int first ) {
    vlc.setRange( first, _beforeCount, _afterCount );
    return getPage();
}
boolean getEntries() {
    // Specify necessary controls for vlv
    if (_pageControls != null) {
        try {
            LDAPSearchConstraints cons =
                ldc.getSearchConstraints();
            cons.setServerControls( ldc.SERVERCONTROLS,
                _pageControls );
        } catch ( LDAPException e ) {
            System.out.println( e + ", setting vlv control" );
        }
    }
    // Empty the buffer
    _entries.removeAllElements();
    // Do a search
    try {
        String[] attrs = {"cn"};
        LDAPSearchResults result =
            _ldc.search( base,
                LDAPConnection.SCOPE_SUB,
                filter,
                attrs,
                false,
                cons );
        while ( result.hasMoreElements() ) {
            LDAPEntry entry = (LDAPEntry)result.nextElement();
            LDAPAttribute attr = entry.setAttribute( attrs[0] );
            if ( attr != null ) {
                Enumeration en = attr.getStringValues();
                while ( en.hasMoreElements() ) {
                    String name = (String)en.nextElement();
                    _entries.addElement( name );
                }
            }
        }
    }
    Expires October 2001
    [Page 22]
```
catch ( LDAPException e ) {
    System.out.println( e + ", searching" );
    return false;
}

return true;

// Fetch a buffer
boolean getPage() {
    // Get the actual entries
    if ( !getEntries() )
        return false;
    // Check if we have a control returned
    LDAPControl[] c = _ldc.getResponseControls();
    LDAPVirtualListResponse nextCont = null;
    if ( c != null ) {
        for( int i = 0; i < c.length; i++ ) {
            if ( c[i] instanceof LDAPVirtualListResponse ) {
                nextCont = (LDAPVirtualListResponse)c[i];
                break;
            }
        }
    }

    if ( nextCont != null ) {
        _selectedIndex = nextCont.getFirstPosition() - 1;
        _top = Math.max( 0, _selectedIndex - _beforeCount );
        // Now we know the total size of the virtual list box
        _size = nextCont.getContentCount();
        _vlc.setListSize( _size );
    } else {
        System.out.println( "Null response control" );
    }
    return true;
}

// Called by JList to fetch data to paint a single list item
public Object getElementAt(int index) {
    if ( (index < _top) || (index >= _top + _entries.size()) ) {
        getPage( index );
    }
    int offset = index - _top;
    if ( (offset < 0) || (offset >= _entries.size()) )
        return new String( "No entry at " + index );
    else
        return _entries.elementAt( offset );
}

// Called by application to find out the virtual selected index
public int getSelectedIndex() {
    return _selectedIndex;
}

// Called by application to find out the top of the buffer
public int getFirstIndex() {

return _top;
}
public void setPageSize( int size ) {
    _pageSize = size;
}
Vector _entries = new Vector();
protected boolean _initialized = false;
private int _top = 0;
protected int _beforeCount;
protected int _afterCount;
private int _pageSize = 10;
private int _selectedIndex = 0;
protected LDAPControl[] _pageControls = null;
protected LDAPVirtualListControl _vlc = null;
protected int _size = -1;
private String _base;
private String _filter;
private LDAPConnection _ldc;
}
Starting a persistent search

```java
import org.ietf.ldap.*;
import org.ietf.ldap.controls.*;
import java.util.*;

public class PersistSearch implements Runnable{

    public PersistSearch() {
    }

    public static void main(String[] argv) {
        Thread th = new Thread(new PersistSearch(), "conn");
        th.start();
        System.out.println("Main thread, waiting for " + 
            "some action");
    }

    public static void printResults(String str, 
        LDAPSearchResults myResults) {
        LDAPEntry myEntry = null;

        /* hasMoreElements() will block until there is a change 
            on the server satisfying our search conditions. When
            it returns, we can see what has changed. The loop is 
            then repeated, and hasMoreElements() will block again 
            until there are additional changes on the server. */
        while ( myResults.hasMoreElements() ) {
            /* A new Richard has appeared, let’s get his 
               attributes */
            System.out.println("**** " + str + "****");
            try {
                myEntry = myResults.next();
            } catch (LDAPReferralException e) {
                /* Or was it a referral? */
                LDAPUrl[] urls = e.getURLs();
                System.out.println("Referral received:");
                for (int i = 0; i < urls.length; i++)
                    System.out.println("  " + urls[i].getUrl());
            }
            String nextDN = myEntry.getDN();
            System.out.println( nextDN );
            LDAPAttributeSet entryAttrs = myEntry.getAttributeSet();
            Enumeration attrsInSet = entryAttrs.getAttributes();
            while ( attrsInSet.hasMoreElements() ) {
                LDAPAttribute nextAttr =
                    (LDAPAttribute)attrsInSet.nextElement();

                String attrName = nextAttr.getName();
                System.out.println("\t" + attrName + ":");
                Enumeration valsInAttr = nextAttr.getStringValues();
                while ( valsInAttr.hasMoreElements() ) {
```

Expires October 2001
String nextValue =
  (String)valsInAttr.nextElement();
  System.out.println( "\t\t" + nextValue );
}
System.out.println("\n");
}

public void run() {
    /* Connect to standard port on local host */
    String hostname = "localhost";
    int portnum = 389;

    /* We want to be notified when any Richard is added to
        any part of the directory under "o=Airius.com".
        We’re not interested in any Richards already there.
        We also don’t care for any return change controls.
        We only want to do this search if the server
        supports persistent search, so set isCritical to
        true.
        When a Richard is added, we want to know his email
        address. */
    String filter = "givenname=Richard";
    String searchbase = "o=Airius.com";
    int scope = LDAPConnection.SCOPE_SUB;
    String[] attrs = {"mail"};
    int op = LDAPPersistSearchControl.ADD;
    boolean changesOnly = true;
    boolean returnControls = false;
    boolean isCritical = true;

    try {
        /* Connect */
        LDAPConnection ld = new LDAPConnection();
        ld.connect(hostname, portnum);

        LDAPSearchConstraints cons = ld.getSearchConstraints();
        cons.setBatchSize(1);
        LDAPPersistSearchControl control =
            new LDAPPersistSearchControl( op, changesOnly,
                                          returnControls,
                                          isCritical );
        cons.setServerControls( control );

        /* The call to search will return almost immediately */
        LDAPSearchResults res = ld.search( searchbase,
                                            scope,
                                            filter, attrs,
                                            false, cons );
        printResults("Persistent Search ", res);
    } catch (Exception e) {
System.out.println(e.toString());
}
}

9. Appendix B - Changes from draft-ietf-rweltman-ldap-java-controls-06.txt

LDAPSortResponse and LDAPSortControl
Added the response control to represent a server’s response to a sort control. Previously the LDAPSortControl was returned as a response control as well.

LDAPAuthRequestControl, LDAPAuthResponse, LDAPDuplicateEntryControl, LDAPDuplicateEntryResponse
Added these controls.

10. Appendix C - Changes from draft-ietf-rweltman-ldap-java-controls-05.txt

LDAPSortKey
Defined this control, since it is no longer defined in [2].

11. Appendix D - Changes from draft-ietf-rweltman-ldap-java-controls-03.txt

LDAPEntryChangeControl, LDAPProxiedAuthControl
Added these controls.

LDAPVirtualListControl
Added constructors that take a context argument, and getContext() and setContext().

LDAPVirtualListResponse
Removed parseResponse(). Added getContext().

Expires October 2001
Removed parseResponse(). Added getFailedAttribute() and getResultCode().

Appendix

Updated examples for changed LDAPVirtualResponse and LDAPSortControl API.