Introduction

While there is generally no need for programming to make use of the Layer2 Business Data List Connector (BDLC) features to integrate external data with SharePoint, there is a BDLC API available. By using the BDLC API, it is possible to access features and settings of the tool in .NET code projects.

With the help of the API you can:

- Connect SharePoint lists with external data sources, provide all required settings
- Apply custom column/field mappings
- Manage initial write-backs (SharePoint to external source)
- Update connected lists with external data
- Maintain lists, release list locks, repairing event receivers
- Encrypt or decrypt security-relevant settings
- Disconnect SharePoint lists from external data sources

You can also make use of the SharePoint API to manage other tasks like:

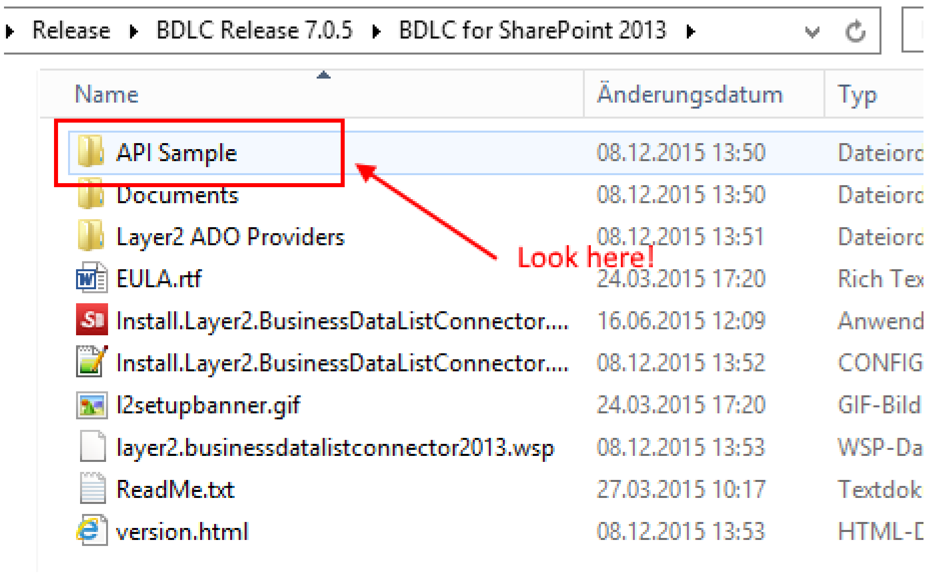
- Create SharePoint lists
- Create specific SharePoint columns
- Remove SharePoint lists

Both of the APIs, SharePoint and Layer2 BDLC, together allow for the managing of all tasks usually required to automate the provisioning and execution of connections.

The next section will provide some examples about how to use the BDLC API for the most common scenarios. In addition, please see the API Reference below for more details.

How to Reference the BDLC Assembly

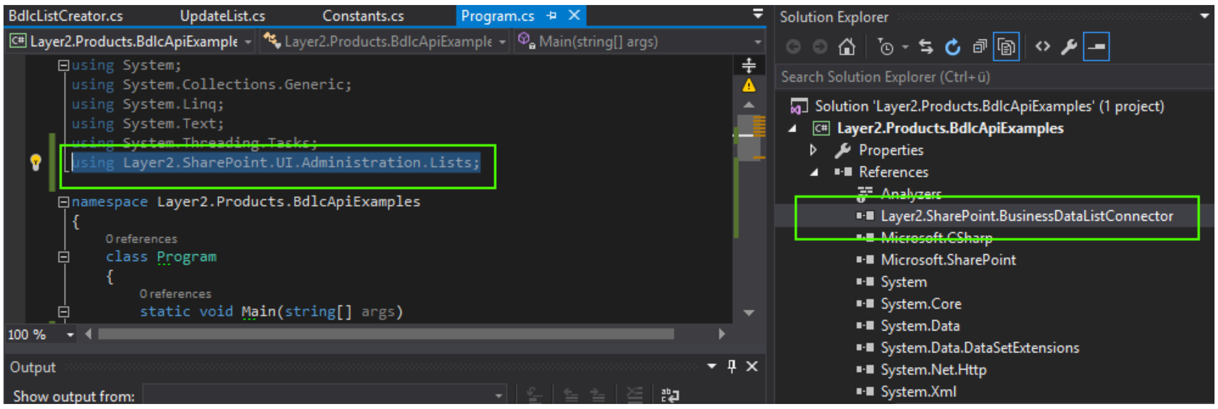
The BDLC assembly can be found in the installation package under API Sample/Layer2.SharePoint.BusinessDataListConnector.dll.



Name	Änderungsdatum	Typ
API Sample	08.12.2015 13:50	Dateiord
Documents	08.12.2015 13:50	Dateiord
Layer2 ADO Providers	08.12.2015 13:51	Dateiord
EULA.rtf	24.03.2015 17:20	Rich Tex
Install.Layer2.BusinessDataListConnector....	16.06.2015 12:09	Anwend
Install.Layer2.BusinessDataListConnector....	08.12.2015 13:52	CONFIG
l2setupbanner.gif	24.03.2015 17:20	GIF-Bild
layer2.businessdatalistconnector2013.wsp	08.12.2015 13:53	WSP-Da
ReadMe.txt	27.03.2015 10:17	Textdok
version.html	08.12.2015 13:53	HTML-D

Figure 1 - Location of the API assembly

After referencing, the using-namespace will be *Layer2.SharePoint.UI.Administration.Lists*.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Layer2.SharePoint.UI.Administration.Lists;

namespace Layer2.Products.BdlcApiExamples
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
```

```
Solution 'Layer2.Products.BdlcApiExamples' (1 project)
  Layer2.Products.BdlcApiExamples
    Properties
    References
      Analyzers
      Layer2.SharePoint.BusinessDataListConnector
      Microsoft.CSharp
      Microsoft.SharePoint
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Net.Http
      System.Xml
```

Figure 2 - Example of referencing the API in a Visual Studio project

Code Examples

Included in the BDLC installation package is a fully functional c#.net project. This is a Visual Studio 2015 console project with a referenced SharePoint 2013 assembly. However, you can use other project types as well. It has been implemented into SharePoint projects, in web services, or in WinForm projects.

Feel free to make use of the examples in your own projects. The code is provided as-is. Some of the included examples do not use the BDLC API, but show best-practices to use when coding against the BDLC in SharePoint.

If you need to access the BDLC Configuration List directly, it is recommended that you use the `BdlcConfigList` class in the example project. This is a downsized version of the code that is similar to what is used in BDLC.

Enabling Lists with BDLC

To connect a list with an external data source, the *ListSettings* object has to be created with the key features of this list as shown in the parameter list of the method below.

If you do not want to enable the background update, the parameters `Interval` and `NextRun` can be ignored.

```
public static void Enable(SPWeb web, SPList list, bool enableWriteBack, string connectionstring, string
connectionProvider, string selectStatement, string primaryKey, string interval, DateTime nextRun)
{
        ListSettings listSettings = new ListSettings
        {
                ConnectionString = connectionstring,
                SelectStatement = selectStatement,
                UseWriteBack = enableWriteBack,
                Provider = connectionProvider,
                PrimaryKeys = primaryKey,
                WebId = web.ID,
                ListId = list.ID,
                Interval = interval,
                NextRun = nextRun
        };

        var businessDataListConnectorLogic = new BusinessDataListConnectorLogic();
        businessDataListConnectorLogic.Init(web, list, listSettings);
        businessDataListConnectorLogic.ExpandList();
        businessDataListConnectorLogic.SaveSettings();
        businessDataListConnectorLogic.EnableOrDisableWriteBack(listSettings.UseWriteBack);
}
```

After configuring the *ListSettings* object, the BDLC logic class is initialized. ExpandList() will create all columns in the list with auto-mapping. Please note that some columns are not automatically able to be mapped, like column names with several umlauts, whitespaces, or longer than 32 characters.

With SaveSettings, the connection is stored to the BDLC Configuration List. If no record exists in this list with a matching key (web ID and list ID), then it will be created.

EnableOrDisableWriteBack will now set up the event receivers for *WriteBack*.

Using Custom Mapping

If a custom mapping is needed, BDLC Configuration List field “MappingTable” has to be used.

The mapping is expressed by a XML value with the following schema:

```
<fieldMappings autoMapping="False">
  <fieldMapping external="my id" internal="my_x0020_id" internaldisplay="my id" ignore="false" />
  <fieldMapping external="Value001" internal="OtherValue" internaldisplay="Value 001" ignore="false" />
  <fieldMapping external="Value002" internal="Value002" internaldisplay="Value002 2" ignore="false" />
  <fieldMapping external="id1" internal="?" internaldisplay="?" ignore="true" />
</fieldMappings>
```

All defined fields are mapped, all other fields are handled as non-present to BDLC. A configured field set to “ignore=true” will not be processed by the **Add Columns** button in the BDLC UI or the ExpandList() method via API.

Initial Write-back

This method will write back data from an existing and filled SharePoint list to your connected data source. Only matching columns will be handled. Make sure your data source is configured properly and provides a primary key for identification.

Note: This will also work if the data source table is populated already with data. Then it will overwrite any existing data with matching primary keys.

```
public static string DoInitialWriteback(SPList list)
{
   var logic = new BusinessDataListConnectorLogic();
   logic.Init(list);
   return logic.InitialWriteback();
}
```

Note: This method is intended for one-off and special use-cases on a list. It is NOT recommended that you use this on a regular basis on a single list.

Updating Lists with BDLC

The core advantage of using the BDLC API is the ability to trigger updates on BDLC connected lists on your own behalf, with your own conditions or individual timing. BDLC updates can be included with your own event receivers, custom actions in the ribbon, sites, or into workflows.

Update a Single List

Updating a single list is a call to the update command after initializing the BDLC logic:

```
public static string UpdateThisList(SPList list)
{
        var logic = new BusinessDataListConnectorLogic();
        logic.Init(list);
        return logic.UpdateList();
}
```

Update All Lists

Another possibility is to loop through all items in the BDLC Configuration List and run the Update on each item. Here one can add additional filters to exclude certain lists.

```
public static List<string> UpdateAllListsInSiteCollection(SPSite site)
{
        List<string> results = new List<string>();
        SPList configurationList = BdlcConfigList.GetList(site.RootWeb);

        foreach (SPListItem configItem in configurationList.Items)
        {
                 bdlcWeb = site.OpenWeb(new Guid(configItem[Constants.WEB_ID].ToString()));
                SPList bdlcList = bdlcWeb.Lists[new Guid(configItem[Constants.LIST_ID].ToString())];
                results.Add(UpdateThisList(bdlcList));
        }
        return results;
}
```

Simulate Background Update

The third option is to simulate the background update. This method will loop through the BDLC Configuration List, but the update will only be performed on lists matching the requirements for a current background update. So, if no **Interval** or **Next Run Date** is configured in the BDLC Configuration List or the next run is still in the future, this method will skip the update.

A conceivable use could be a simulation of a timer job, if you do not want to make use of the BDLC timer job due to a very large farm with thousands of site collections or if you want to set up custom timer jobs to run certain lists.

```
public static void PerformBackgroundUpdate(SPSite site)
{
        var logic = new BusinessDataListConnectorLogic();
        logic.UpdateAllLists(site.RootWeb);

}
```

Maintaining Lists

The following section includes some common scenarios for maintaining BDLC connected lists.

Release List Lock

Normally, BDLC should always release a lock on a list whether the action on this list was successful or not. Nonetheless, onsome occasions the BDLC is unable to release the lock, e.g. if the server restarts during an update or the w3wp process crashes.

This method checks all items in the BDLC Configuration List for invalid seeming locks and releases them. Please use only if there is a strong suspicion that invalid locks are present to avoid possible inconsistencies with running updates.

```
internal static string MaintainConnectionLocks(SPSite site) //other possible parameter is the rootweb
{
        var businessDataListConnectorLogic = new BusinessDataListConnectorLogic();
        return businessDataListConnectorLogic.EnsureGeneralUnlock(site);
}
```

You can target this also on a single list:

```
internal static void Unlock(SPList list)
{
        var logic = new BusinessDataListConnectorLogic();
        logic.Init(list);
        logic.EnsureUnlock();
}
```

Repair Event Receivers

On some rare occasions, the event receivers are broken or removed by SharePoint. The method will check for all events that should be bound to the list and reattaches them. Despite the naming it will check for all used events:

- Write-back events (if configured)
- Workflow starter events to start declarative workflows after a timer job update
- List deleted event when a connected list is deleted but the BDLC Configuation Item still exists.

```
internal static void RepairWriteback(SPSite site)
{
        BusinessDataListConnectorLogic.RepairWriteBackOnAllLists(site);
}
```

Use API Calls from a Web Service or 3rd-Party Website

BDLC checks on the write-back event if the action is started from certain sources. This is necessary due to the handling of date/time objects and other event properties in the SharePoint events. If BDLC API is called from a website or a web service, you have to inform BDLC that these addresses are also valid like the BDLC settings page or the update page.

To do so, use this method and acknowledge BDLC with the web URL calling the BDLC. Normally this is the .asmx file name of the used web service.

```
internal static void RegisterWebServiceAndUpdateListInBdlc(SPSite site, Guid listId, Guid webId)
{
        BusinessDataListConnectorLogic logic = new BusinessDataListConnectorLogic();
        SPWeb web = site.OpenWeb(webId);
        SPList list = web.Lists[listId];
        logic.Init(web, list);
        logic.RegisterCustomUrl("BdlcService.asmx"); // <= this is an example value, insert your
webservice here!
        string result = logic.UpdateList();
}
```

Update Sensitive Data in the BDLC Configuration List

BDLC stores the properties `ConnectionString` and `SelectStatement` in an encoded state to prohibit access to sensitive data like passwords or login credentials.

This method shows how to update encoded values directly in the BDLC Configuration List.

```
internal void UpdateBdlcListConnectionItem(SPList bdlcConfigList, SPList listToUpdate,
            string provider, string connectionString, string selectStatement)
{
        string queryString =
                "<Where><And><Eq><FieldRef Name='{0}' /><Value
                Type='Text'>{1}</Value></Eq><Eq><FieldRef Name='{2}' /><Value
                Type='Text'>{3}</Value></Eq></And></Where><OrderBy><FieldRef Name='ID' /></OrderBy>";
        queryString = String.Format(queryString, "bdlcListId", listToUpdate.ID, "bdlcWebId",
        listToUpdate.ParentWeb.ID);
        SPQuery query = new SPQuery { Query = queryString };

        var results = bdlcConfigList.GetItems(query);
        if (results == null || results.Count == 0)
        return;

        string encryptedConnectionString = RijndaelCiphering.Encrypt(connectionString);
        string encryptedselectStatement = RijndaelCiphering.Encrypt(selectStatement);

        SPListItem item = results[0];
        item[Constants.SELECT_STATEMENT] = encryptedselectStatement;
        item[Constants.CONNECTION_PROVIDER] = provider;
        item[Constants.CONNECTION_STRING] = encryptedConnectionString;
        item.Update();
}
```

Note: From BDLC version 7.1.0.0 forward, it is possible to use plain text in `ConnectionString` or `SelectStatement`.

Disabling BDLC on Lists

If a list needs to be disconnected from BDLC, this method will disable the attached events and then delete the matching item in the BDLC Configuration List.

```
internal static void DisableBdlcInThisList(SPList list)
{
   DisableListEvents(list);
   SPList configurationList = BdlcConfigList.GetList(list.ParentWeb);
   SPListItem configItem = CheckIfListsExistsInConfigList.GetConfigItem(configurationList, list);
   configurationList.GetItemById(configItem.ID).Delete();
}
```

API Reference

BusinessDataListConnectorLogic Class

Most of the API functionality can be found in the class BusinessDataListConnectorLogic.

Referenced version of BDLC: 7.3.0.1 (May 2016)

Init(SPList)

This will initialize the logic object with the given BDLC-enabled list, its parent web, and load the given configuration from the BDLC Configuration List. Use this only for lists that are already enabled as a BDLC connection.

Namespace:

```
        Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
        public void Init(SPList list)
```

Parameter:

list: your *SPList* object

Returns:

-

Example:

```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
```

Init(SPWeb, SPList)

This will initialize the logic object with the given BDLC-enabled list, given web, and load the given configuration from the BDLC Configuration List. Use this only for lists that are already enabled as a BDLC connection.

(**Note:** This is deprecated but still present due to backward compatibility)

Namespace:

```
        Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
        public void Init(SPWeb web, SPList list)
```

Parameter:

web: the web in which the new list exists

list: your *SPList* object

Returns:

-

Example:

-

Init(SPWeb, SPList, ListSettings)

Use this variant of *Init* for creating or enabling new BDLC connections on a list. You have to configure the *ListSettings* object before passing it. This method is for use with adding new BDLC-enabled lists and manually handing over the list settings.

Namespace:

```
        Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
        public void Init(SPWeb web, SPList list, ListSettings settings)
```

Parameter:

Web: the web in which the new list exists
list: your *SPList* object
settings: The list settings, see ListSettings class

Returns:

-

Example:

```
var businessDataListConnectorLogic = new BusinessDataListConnectorLogic();
businessDataListConnectorLogic.Init(web, list, listSettings);
```

InitialWriteback()

This method will write back data from an existing and filled SharePoint list to your connected data source. Only matching columns will be handled. Make sure your data source is configured properly and provides a primary key for identification. If the data source table is populated already with data, matching data **will be overwritten** by the SharePoint content. Non-matching data remain as-is.

Namespace:

```
Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
public string InitialWriteback()
```

Parameter:

-

Returns:

String result: If everything works fine: *String.Empty*. OTHERWISE, the error message

Example:

```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
return logic.InitialWriteback();
```

EnsureUnlock()

Ensures that the BDLC Configuration List entry for the initialized list is set to "Lock=false". Lists marked as locked cannot be processed by BDLC.

Namespace:

```
Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
public void EnsureUnlock()
```

Parameter:

-

Returns:

-

Example:

```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
logic.EnsureUnlock();
```

EnsureUnlock(bool)

Lock or unlock lists. Locked lists cannot be processed by BDLC. If *Writeback* is enabled, also users cannot edit items in a locked list.

Namespace:

```
Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
public void EnsureUnlock(bool enable)
```

Parameter:

enable: TRUE if list should be locked, otherwise FALSE

Returns:

-

Example:

```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
logic.EnsureUnlock(true); // list is locked now
```

EnsureGeneralUnlock (SPWeb)

This method retrieves the configuration list of the current site collection, loops through all entries and releases all locks. Handle with care: it is recommended that you only use this directly before or after system restarts, or timer job recycling events. Otherwise, this could cause inconsistent data.

Note: Use this method without any initialization (see example).

Namespace:

```
Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
public string EnsureGeneralUnlock(SPWeb web)
```

Parameter:

web: Any web of the target site collection

Returns:

A status message like “EnsureGeneralUnlock() successfully unlocked <2> items.”

Example:

```
var logic = new BusinessDataListConnectorLogic();
result = logic.EnsureGeneralUnlock(web);
```

EnsureGeneralUnlock (SPSite)

This method retrieves the BDLC configuration List of the current site collection, loops through all entries and releases all locks.

Handle with care: it is recommended that you only use this directly before or after system restarts, or timer job recycling events. Otherwise, this could cause inconsistent data.

Note: Use this method without any initialization (see example).

Namespace:

```
Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
public string EnsureGeneralUnlock(SPSite site)
```

Parameter:

site: your *SPSite* object

Returns:

A status message like “EnsureGeneralUnlock() successfully unlocked <2> items.”

Example:

```
var logic = new BusinessDataListConnectorLogic();
result = logic.EnsureGeneralUnlock(site);
```

EnableOrDisableWorkflowActivation (bool)

Enables or disables the automatic starting of declarative workflows for the connected list in case of timer job updates.

Namespace:

`Layer2.SharePoint.UI.Administration.Lists`

Syntax:

`public void EnableOrDisableWorkflowActivation(bool enable)`

Parameter:

enable: TRUE if this event should be enabled, otherwise FALSE

Returns:

-

Example:

```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
logic.EnableOrDisableListEvents(true);
```

EnableOrDisableWriteBack (bool)

Enables or disables the *Writeback* from SharePoint to the external data source for the connected list in case of changes or updates on this list.

Namespace:

`Layer2.SharePoint.UI.Administration.Lists`

Syntax:

`public void EnableOrDisableWriteBack(bool enable)`

Parameter:

enable: TRUE if this event should be enabled, otherwise FALSE

Returns:

-

Example:

```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
logic.EnableOrDisableWriteBack(true);
```

EnsureWriteBackOnAllLists (SPSite)

Checks all in the BDLC configuration List present connections for proper enabled *Writeback* event receivers, if *Writeback* was checked.

Namespace:

`Layer2.SharePoint.UI.Administration.Lists`

Syntax:

`public void EnsureWriteBackOnAllLists(SPSite site)`

Parameter:

site: the current *SPSite*

Returns:

-

Example:
```
var logic = new BusinessDataListConnectorLogic();
logic.EnsureWriteBackOnAllLists(site);
```

EnableOrDisableListEvents (bool)

Adds or deletes the BDLC event receiver like *Writeback* or automatic workflow activation. This is a combination of *EnableOrDisableWriteBack* and *EnableOrDisableWorkflowActivation*.

Namespace:

    Layer2.SharePoint.UI.Administration.Lists

Syntax:

    public void EnableOrDisableListEvents(bool enable)

Parameter:

enable: TRUE if the events should be enabled, otherwise FALSE

Returns:

-

Example:
```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
logic.EnableOrDisableListEvents(true);
```

ExpandList ()

Reads column information from data source and expands your BDLC list with appropriate *SPList* columns. Use this to add columns when setting up a list for BDLC or when it is necessary to update the list with new columns.

Namespace:

    Layer2.SharePoint.UI.Administration.Lists

Syntax:

    public void ExpandList()

Parameter:

-

Returns:

-

Example:
```
var businessDataListConnectorLogic = new BusinessDataListConnectorLogic();
businessDataListConnectorLogic.Init(web, list, listSettings);
logic.ExpandList();
```

LoadSettings ()

Loads the BDLC configuration data for an existing and configured list and returns it encapsulated as a *ListSettings* object.

Namespace:

    Layer2.SharePoint.UI.Administration.Lists

Syntax:

    public ListSettings LoadSettings()

Parameter:

-

Returns:

listSettings: see *ListSettings* for more details

Example:

```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
ListSettings listSettings = logic.LoadSettings();
```

SaveSettings()

Saves changed or new settings to the BDLC Configuration List.

Namespace:

```
Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
public void SaveSettings()
```

Parameter:

-

Returns:

-

Example:

```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
// do sth
logic.SaveSettings();
```

SaveSettings (ListSettings, bool)

This method is deprecated. Please use *SaveSettings*() instead.

Namespace:

```
Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
public void SaveSettings(ListSettings listSettings, bool createList)
```

Parameter:

listSettings: ListSettings object with configuration data

createList: flag if the list exists or not

Returns:

-

Example:

```
logic.SaveSettings(listSettings, true);
```

RegisterCustomUrl (string)

When using the BDLC API from within a web service, the write back could throw errors and the update would then be aborted. The API now provides a method to register the web service to BDLC. It is best to register the .asmx address of your service.

This method receives a single service address.

Namespace:

```
Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
public void RegisterCustomUrl(string url)
```

Parameter:

url: the URL of the webservice to register

Returns:

-

Example:
```
var logic = new BusinessDataListConnectorLogic();
logic.RegisterCustomUrl(url);
```

RegisterCustomUrl (List<string>)

When using the BDLC API from within a web service, the write back could throw errors and the update would then be aborted. The API now provides a method to register the web service to BDLC. It is best to register the .asmx address of your service.

This method receives a list of service addresses.

Namespace:
```
Layer2.SharePoint.UI.Administration.Lists
```
Syntax:
```
public void RegisterCustomUrl(List<string> urls)
```
Parameter:

urls: a list of URLs to register

Returns:

-

Example:
```
var logic = new BusinessDataListConnectorLogic();
logic.RegisterCustomUrl(listOfUrls);
```

UpdateList ()

Starts the synchronization of the connected list. Possible errors are logged in the returning result string.

Namespace:
```
Layer2.SharePoint.UI.Administration.Lists
```
Syntax:
```
public string UpdateList()
```
Parameter:

-

Returns:

A string with update information similar to the settings page of the BDLC-enabled list

Example:
```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
result = logic.UpdateList();
```

UpdateList (ListSettings)

Starts the synchronization of the connected list. Possible errors are logged in the returning result string.

Note: This method is deprecated and only present due to compatibility reasons.

Namespace:
```
Layer2.SharePoint.UI.Administration.Lists
```
Syntax:
```
public string UpdateList(ListSettings listSettings)
```

Parameter:

listSettings: a fully configured *ListSettings* object

Returns:

A string with update information similar to the settings page of the BDLC-enabled list

Example:

```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
result = logic.UpdateList(listSettings);
```

UpdateListNotSilenced ()

Starts the synchronization of the connected list.

Note: This method will throw errors if something unexpected happens.

Namespace:

```
Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
public string UpdateListNotSilenced()
```

Parameter:

-

Returns:

A string with update information similar to the settings page of the BDLC-enabled list

Example:

```
var logic = new BusinessDataListConnectorLogic();
logic.Init(list);
result = logic.UpdateListNotSilenced();
```

RepairWriteBackOnAllLists(SPSite)

Deletes and re-binds all event receivers on all lists in this site collection.

Namespace:

```
Layer2.SharePoint.UI.Administration.Lists
```

Syntax:

```
public static void RepairWriteBackOnAllLists(SPSite site)
```

Parameter:

site: The *SPSite* object the BDLC is activated in.

Returns:

-

Example:

```
BusinessDataListConnectorLogic.RepairWriteBackOnAllLists(site);
```

ListSettings Class

ListSettings is a container to hold all the connection properties.

Constructors

Recommended:

```
public ListSettings()
```

Deprecated:

```
public ListSettings(string connectionString,
             string provider,
             string selectStatement,
             string primaryKeys,
             Guid webId,
             Guid listId,
             bool hasTimerJob,
             bool sync,
             bool locked,
             string timeOut,
             string interval,
             DateTime nextRun)
```

```
public ListSettings(string connectionString,
            string provider,
            string selectStatement,
            string primaryKeys,
            Guid webId,
            Guid listId,
            bool hasTimerJob,
            string timeOut,
            bool sync,
            bool locked,
            bool allowManualUpdates,
            int interval,
            DateTime nextRun,
            DateTime lastRun,
            string lastDuration,
            string lastMessage)
```

For internal use:

```
public ListSettings(BdlcConfigurationListItem configItem)
```

Properties

```
public string ConnectionString { get; set; }
```

The given connection string for this list. This value is encrypted by BDLC. From version 7.1.0.0 forward, you
can use unencrypted values as well, e.g. for provisioning websites with pre-configured BDLC settings.

```
public string Provider { get; set; }
```

The .net name of the used connection provider for this list.

```
public string SelectStatement { get; set; }
```

The given select statement (if present) for this list. This value is encrypted. From version 7.1.0.0 forward,
you can use unencrypted values as well, e.g. for provisioning websites with pre-configured BDLC settings.

```
public string PrimaryKeys { get; set; }
```

The replication key used by BDLC to ensure the data mapping. Use with <,> or <;> separator.

```
public Guid WebId { get; set; }
```

The ID (GUID) of the SharePoint web containing the list.

`public Guid ListId { get; set; }`
The ID (GUID) of the SharePoint list.

`public bool HasTimerJob { get; set; }`
A Boolean value indicating if this connection uses the background update.
Note: This value will not be saved in the BDLC Configuration List item.

`public bool UseWriteBack { get; set; }`
A true/false value indicating if write back is configured (TRUE) or not (FALSE).

`public bool Sync`
DEPRECATED: Use “UseWriteBack” instead.
A true/false value indicating if write back is configured (TRUE) or not (FALSE).
Note: This value will not be saved in the BDLC Configuration List item.

`public bool AllowManualUpdates { get; set; }`
A Boolean value indicating if a connected list shows the “Update BDLC” ribbon button.
Note: This value will not be saved in the BDLC Configuration List item.

`public bool Locked { get; set; }`
A true/false value. During an update, BDLC sets the lock on the handled list to TRUE. Normally set back to FALSE after an update, whether it succeeds or not.

`public string TimeOut { get; set; }`
The timeout value in seconds used for connections to the external database. Not all providers support a timeout. The default value is 30 seconds.
Note: This value will not be saved in the BDLC Configuration List item.

`public string Interval { get; set; }`
The interval in minutes for the background update if configured. The default value is 60 minutes, the minimum value should be 15 minutes

`public DateTime NextRun { get; set; }`
The scheduled date and time for the next background update.

`public string LastMessage { get; set; }`
The result message of the last manual or background update.
Note: This value will not be saved in the BDLC Configuration List item.

`public string LastDuration { get; set; }`
The duration of the last update (manual or background) in seconds.
Note: This value will not be saved in the BDLC Configuration List item.

`public DateTime LastRun { get; set; }`
The date and time of the last successful background update.
Note: This value will not be saved in the BDLC Configuration list item.

`public bool ListCreated;`
An obsolete setting present for downgrade compatibility.
Note: This value will not be saved in the BDLC Configuration List item.

`public bool Empty { get; set; }`
An internal value for identifying empty or non-configured ListSettings objects.

`public int DelayUntilUnlock { get; set; }`
A minute value for automatically releasing existing locks on the list after this amount of time has spent after the given “Next Run” value. Should be at least twice the given Interval. Note that while this option can be used as a workaround for locking issues, it is not generally recommended. It is

better to fix the errors and possible configuration issues that cause the locking.

```
public string EmailAddress { get; set; }
```
Email addresses of the BDLC error reporting recipients.

```
public string EmailSubject { get; set; }
```
The subject of the error reporting email.
Note: This value will not be saved in the BDLC Configuration List item.

```
public string EmailBody { get; set; }
```
Email text for error reporting. Valid and supported placeholders are described in the "Logging and Alerting" section of the Layer2 Business Data List Connector User Documentation PDF.
Note: This value will not be saved in the BDLC Configuration List item.

```
public string MappingTable { get; set; }
```
The XML configuration when using a custom mapping.

RijndaelCiphering Class

In the BDLC Configuration List, the properties for `SelectStatement` and `ConnectionString` are stored not as plain text but as encrypted (with version 7.1.0.0, the possibility was implemented to store unencrypted values in these fields as well).

Encrypt(string)

Encrypts specified plain text using Rijndael symmetric key algorithm and returns a base64-encoded result.

Namespace:

`Layer2.SharePoint.UI.Administration.Lists.Security`

Syntax:

`public static string Encrypt(string plainText)`

Parameter:

plaintext: value to be encrypted

Returns:

Encrypted value formatted as a base64-encoded string

Example:

```
string encryptedValue = RijndaelCiphering.Encrypt(value);
```

Content