# Table of Contents

**Architectural Design**

- Local DNN Codebase Deployment ......................................................... 2
- Remote .NET or Heterogeneous Deployment ........................................... 2

**.NET API Overview** .............................................................................. 2
- Get – Retrieve Information .................................................................... 2
- Post – Send Information ......................................................................... 2
- Delete – Delete Information .................................................................. 3
- Accord LMS Learner Module .................................................................. 3
- LE Utilization ......................................................................................... 3
- Classes and Configuration ..................................................................... 3

**API Methods (.NET interface)** ................................................................. 3
- Connecting to the API ........................................................................... 3
- Getting the Catalog .............................................................................. 3
- Create Catalog Filters ........................................................................ 3
- Get Catalog View without Folders without Filters ............................... 4
- Get Catalog View without Folders with Filters ..................................... 4
- Launching a Learning Event .................................................................. 5
- Show Window ....................................................................................... 5
- Learning Events .................................................................................... 6
- Get Learning Event .............................................................................. 6
- Get Learning Event Details .................................................................. 6
- Get Learning Event Conditions ............................................................ 7
- Get Learning Event Attempts ............................................................... 7
- Get Learning Event Interactions ........................................................... 8
- Set Learning Event Enrollment ............................................................. 8
- Set Learning Event Archive ................................................................. 8
- Attempts and Interaction Tracking ....................................................... 8
- Learner Client Result Refresh ............................................................... 9
Architectural Design
The Accord LMS.NET API provides access to all the LMS Administration module services. Clients can programatically pull learner catalog, tracking and report information from the Administration module into their own custom DNN application, such as a specialized Learner module. The Administration module will continue to directly track all Learner utilization of Learning Events including attempts, dates, scores, etc.

Local DNN Codebase Deployment
Custom application (client) and Administration (server) modules are both running in the same DNN installation. The custom application calls the .NET API DLL directly. All methods, structures and their descriptions are available via Visual Studio Intellisense.

Remote .NET or Heterogeneous Deployment
REST transport protocol (web services) communicates between the custom application (client) and .NET (server) API are also available. Please consult the appropriate documentation.

All methods, structures and their descriptions are available via Visual Studio Intellisense. This is the most exhaustive and up to date information on the .NET API.

A brief overview and some of the methods and structures are provided below.

.NET API Overview
There are three different protocols for interacting with the .NET API. Even though these are .NET API calls, they are modeled and ready for REST implementation.

Get – Retrieve Information
- CatalogView - TreeGrid for Learner > My Plan, Archive or Catalog
- Learning Event Details
- Condition Objects
- Attempt and Interaction Objects
- RoleAward Objects

Post – Send Information
- Learner Catalog Enrollment
- Learning Event Status Change – My Plan to/from Archive
Delete – Delete Information

- Drop Learner Catalog Enrollment

Accord LMS Learner Module

The Accord LMS Learner module now utilizes all required Administration services via the .NET API. This insures that the current and future API functionality is user and field tested on an ongoing basis.

LE Utilization

The .NET API does not provide any attempt or interaction services for LE utilization. The functionality to launch the LE is provided. The Administration module track LE utilization on the server side. See “Posting Attempts and Interactions” below.

Classes and Configuration

Functionality required for the Learner > Classes and Learner > Configuration tabs is not provided in the API.

API Methods (.NET interface)

All current ‘info’ classes are public (not friend or private) within all the Interzoic DLL set because CBO.fill does not work if the classes are not public. Care should be taken to only utilize the public methods and classes in the Interzoic.LMS.API namespace. Interzoic will not be responsible for maintaining any methods or classes from other namespaces.

Connecting to the API

```vbnet
Dim client As New LMSClient()
```

Getting the Catalog

Create Catalog Filters

```vbnet
Dim myFilters As New CatalogFilters()
myFilters.IsVisible = true
myFilters.IsExpired = True
myFilters.IsPublished = True
myFilters.IsArchived = True
myFilters.IsEnrolledby = EnrolledStatus.All
myFilters.SearchCriteria = String.Empty
```
Get Catalog View without Folders without Filters

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    If Not Page.IsPostBack Then
        Try
            Dim client As API.LMSSClient = New API.LMSSClient(Context)
            Dim managerPortalId As Integer = 0 ' Me.PortalId // the portal Id
            Dim managerModuleId As Integer = 449 ' the catalog (manager) module Id
            Dim targetUserId As Integer = 1400 ' the user (from which to get the LEs) with enrollments in the given Administration module
            Dim learnerModuleId As Integer = -1 ' Me.ModuleId // in this case, null is fine // The learner module Id
            Dim tabId As Integer = -1 ' Me.TabId // in this case, null is fine.

            This is the tab from where the LE will be called (to create the launch link), this tabId is used to create links in notifications to point to the launch page
            Dim ignoreConditions As Boolean ' if true links are generated no matter the LE has or not conditions, if false when some conditions are not met, the link is only a show message

            Dim learningEventsList As IList(Of API.LearningEvent) = client.GetLearningEvents(managerPortalId, targetUserId, managerModuleId, learnerModuleId, tabId, ignoreConditions)
            Dim nn As Integer = learningEventsList.Count
        End If
        Catch exc As Exception
            ProcessModuleLoadException(Me, exc)
        End Try
    End Sub

Get Catalog View without Folders with Filters

Dim learningEventsList As IList(Of LearningEvent) = client.GetLearningEvents(PortalId, UserId, ManagerModuleId, LearnerModuleId, TabId, False, myFilters)

Notes:
- The API will reject calls for a 'UserId' that is flagged as deleted or locked.
- CatalogView is a Folder object set to the root of the Catalog which the Learner can see and access.
- Each Folder object has a list of child Folders (FolderObject) and a list of Learning Events (LearningEventObject).
- CatalogView is built of regular classes and collections for the .NET API. Below is an XML representation to demonstrate the structure and content.

Catalog View - XML

<Folders>
    <Folder>
        <FolderId/>
        <FolderName/>
        <ChildFolders>
            <Folder>
                ...
            </Folder>
        </ChildFolders>
    </Folder>
</Folders>
Launching a Learning Event

The LaunchURL property in LearningEventObject is a javascript executable string that has different formats depending on the window type, and prerequisites:

Show Window

- **ShowPopUp(URL, width, height, behaviors)**
  Will launch a popup window with the URL passed as a parameter. Width and height are the dimensions of the defined LE Type, and the settings are the same as used in the javascript function window.open function call.

Here is an example implementation of ShowPopUp function:
function ShowPopUp(URL, width, height, behaviors) {

  //Center Vertically
  
  if (behaviors != "")
    behaviors = "," + behaviors;

  var finalBehaviors = "width=" + width + ",height=" + height + behaviors;
  activeWindow = window.open(URL, '_blank', finalBehaviors);
}

• **ShowDialog(URL, width, height)**
  
  Same as ShowPopUp, but presents a modal popup window.

• **ShowActualWindow(URL)**
  
  Opens the URL in current window.

• **ShowConditionMessage(htmlObjectSender, LEId)**
  
  This function provides a message to the Learner detailing conditions that that are required to view this LE.

These functions must be implemented in the new Learner UI using the appropriate settings. URL is the path that will open the LE. **ShowConditionMessage** requires an appropriate error message
– “Not possible to launch LE, conditions pending.” etc.

**Learning Events**

**Get Learning Event**

Dim LearningEventObject as LearningEvent = Client.GetLearningEventByLEId(LEId, FolderID, UserId, LearnerModuleId, IgnoreConditions)

**Get Learning Event Details**

Dim LearningEventObject as LearningEventDetails = 
Client.GetLearningEventDetailsByLEId(LEId, FolderID, UserId, LearnerModuleId, IgnoreConditions)

Returns all the LE details.
Among other things, will allow displaying this:

![Image of a learner classroom training with details such as title, type, specific, enrolled by, start date, due date, overview/description, expected duration, passing score, target audience, lesson objectives, created date, and updated date.](image)

**Get Learning Event Conditions**

Dim ConditionsMet as IList(Of ConditionInfo)
Dim ConditionsNotMet as IList(Of ConditionInfo)

Dim Condition as Consts.ConditionStatus = Client.GetCondition(PortalId, ManagerModuleId, LEId, UserId, false, ConditionsNotMet, ConditionsMet)

Dim Condition as Consts.ConditionStatus = Client.GetCondition(PortalId, ManagerModuleId, LEId, UserId, false, ConditionsNotMet, ConditionsMet, optional WaiveAttemptLimit, optional WaivePrerequisites, optional WaiveRoleReq)

Check enrollment waivers is used to check or ignore if false the existent enrollment waivers. ConditionsNotMet and ConditionsMet are output parameters, will be filled with the correspondent conditions (both are list of conditions).

All waivers when true ignore the specific type of conditions.

Returns a ConditionStatus, indicating the actual LE status for the user, possible statuses are: None, Required, Satisfied, Limited, NotLimitedYet.

Will allow displaying this:

![Image of a notification message stating that the following conditions or limits apply: disqualifying conditions - prerequisite: learner overview - quiz passed.](image)

**Get Learning Event Attempts**

Dim Attempts As IList(Of AtemptListInfo) = Client.GetAttempts(PortalId, LEId, UserId)
Returns a collection of Attempt Objects.

This will allow presenting the following information:

<table>
<thead>
<tr>
<th>Learning Event</th>
<th>Status</th>
<th>Score</th>
<th>Duration</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>What are Business Ethics - Quiz Results</td>
<td>Failed</td>
<td>40%</td>
<td>00:00:22</td>
<td>3 Mar 2010</td>
</tr>
<tr>
<td>What are Business Ethics - Quiz Results</td>
<td>Failed</td>
<td>40%</td>
<td>00:00:19</td>
<td>3 Mar 2010</td>
</tr>
</tbody>
</table>

**Get Learning Event Interactions**

Dim Attempts As IList(Of InteractionInfo) =

Client.GetInteractionsByAttemptId(AttemptId) For an Attempt object you can request a collection of interactions (answers)

This will allow presenting the following information:

**Set Learning Event Enrollment**

Client.EnrollLE(LEId, PortalId, ManagerModuleId, ParentFolderId, UserId)

Client.DropEnrollLE(LEId, PortalId, ManagerModuleId, ParentFolderId, UserId)

**Set Learning Event Archive**

Client.Archive(LEId, FolderId, UserId, ManagerModuleId, PortalId) Client.UnArchive(LEId, FolderId, UserId, ManagerModuleId, PortalId)

**Attempts and Interaction Tracking**

When a LE has tracking set, the URL used to open the LE points to a loader page. This loader is considered a Administration component, and is in charge to launch the LE content in an environment where it can be tracked. When the LE is a SCORM content (a SCO), this implies a
SCORM API object in available in the page to communicate with the SCO. The loader page will save all the tracking information directly to the Administration database.

**Learner Client Result Refresh**

How does the API inform the Learner Client when to refresh the page to present the completed LE results? If the Learner module uses Telerik Rad Window to implement modal behavior, these are the required steps:

- **Include the JS file RadWindowUtil.js**
  - This JS must be included in the ASCX or ASPX page:
    ```html
    <script type="text/javascript"
    language="javascript">
      if (Telerik.Web.UI.RadWindow) {
        if (!isTelerikCloseRedefined) {
          Telerik.Web.UI.RadWindow.prototype.close = function() {
            window.CustomizedClose();
          }
        }
        var isTelerikCloseRedefined = true;
      }
    </script>
    - **Define the JS function CustomizedClose like this:**
      ```javascript
      function CustomizedClose()
      {
        UnloadAndClose('path/ContentUnload.htm');
      }
      ```
      **NOTE:** The file ContentUnload.htm is provided by us.

- **Define the JS function RefreshTree**
  - NOTE: In this function we do a postback, which allows us to refresh the Tree with the updated data.

*If the Learner module does not use the Telerik Rad Window, these are the required steps:*

- **Define the JS function RefreshTree (will be called by the SCO loader page when it is closed)**
  - NOTE: In this function we do a postback, which allows us to refresh the Tree with the updated data.

**Note:** when a SCO finishes (the SCO calls the terminate function), a customized close() function will be called if exists. If it does not exist, the standard close() will be called. In a normal browser window, close() will cause the window to close.