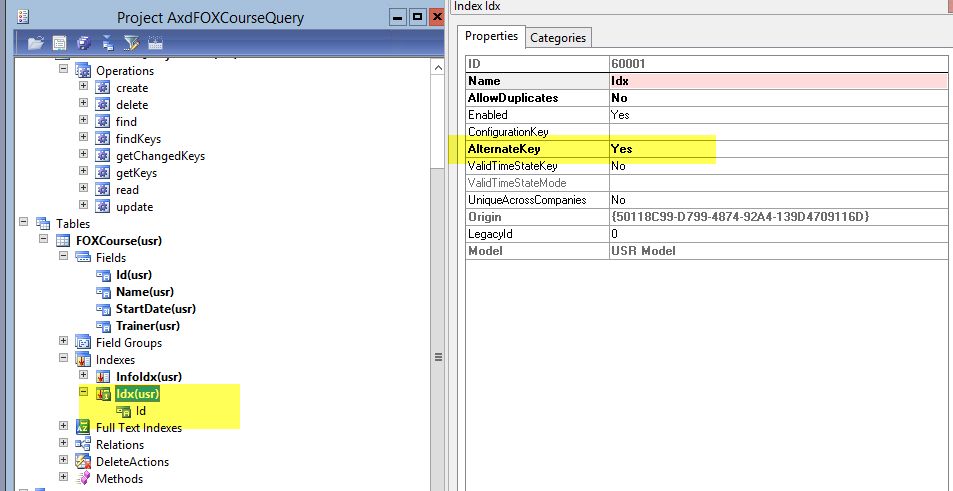
**Perform CRUD operations from Visual Studio console with AIF Document services**

In this post will demonstrate all steps to configure and consume AIF document service for a simple demo table with 4 fields which is not a standard table in Ax. In this table we can store course data.

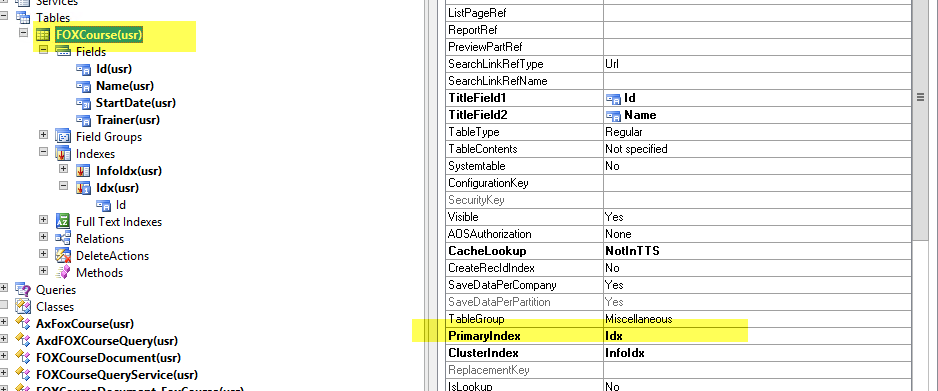
With AIF Document services it's possible to perform CRUD ((Create, Read, Update Delete and Find ) operations to an Ax table from an external system (in our case Visual Studio console) . We will demonstrate all steps from creation of this table , generating AIF Document Service Wizard until the Visual Studio console is ready to do CRUD operations, we are using in this demo.

Our first step is create the table:

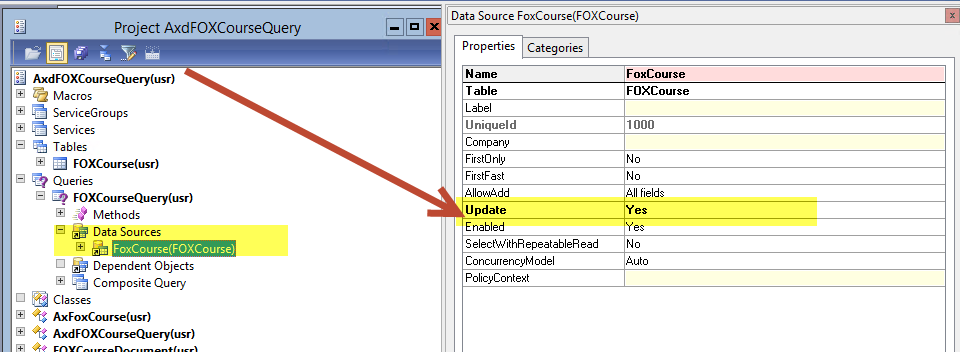
In order to use the Read and Update operator we need to have primary assigned in our table. At index tree in the table we can create primary index by setting index properties AllowDuplicate = "NO" and AlternateKey = "YES".



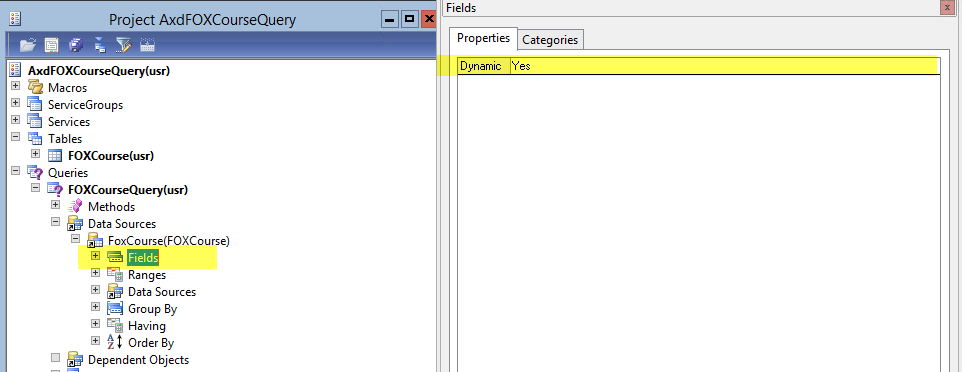
Then at the table properties we need select the primary Index:



Now we will make a AOT query object and use the table to it's datasource. Drag and draw the table to the query datasource. At the datasource properties we should set 'Update' property to 'yes'.

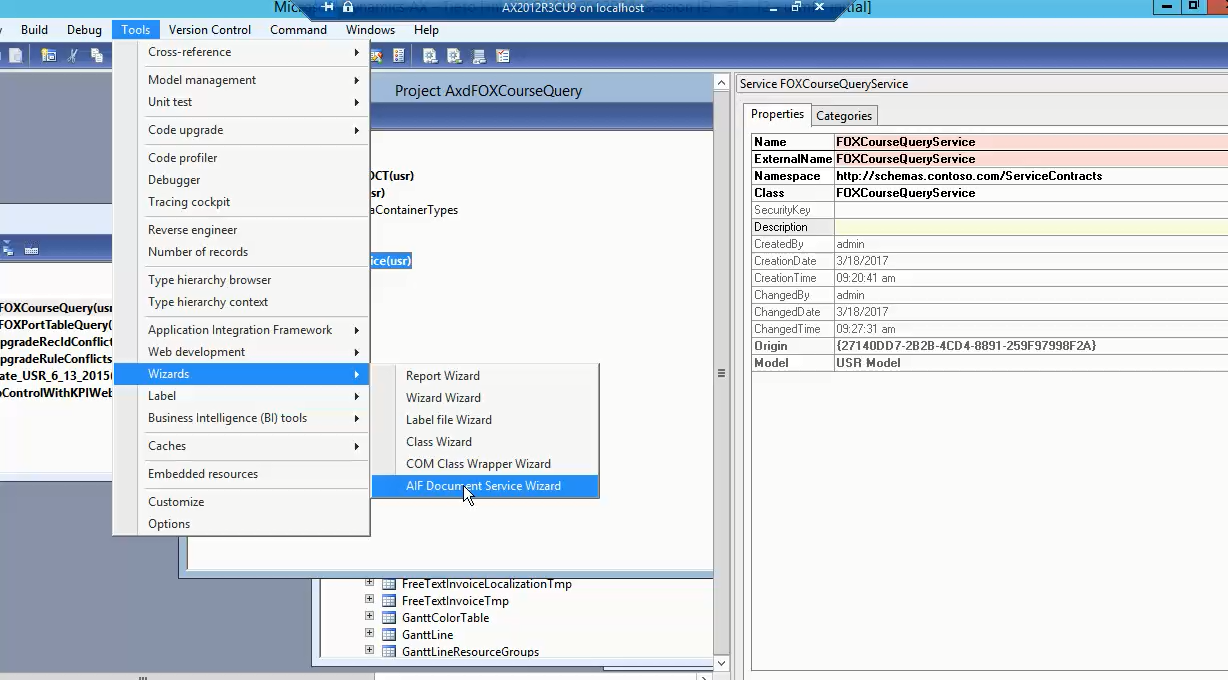


At the 'fields' node of the datasource, the 'Dynamic' property should set to 'Yes'.

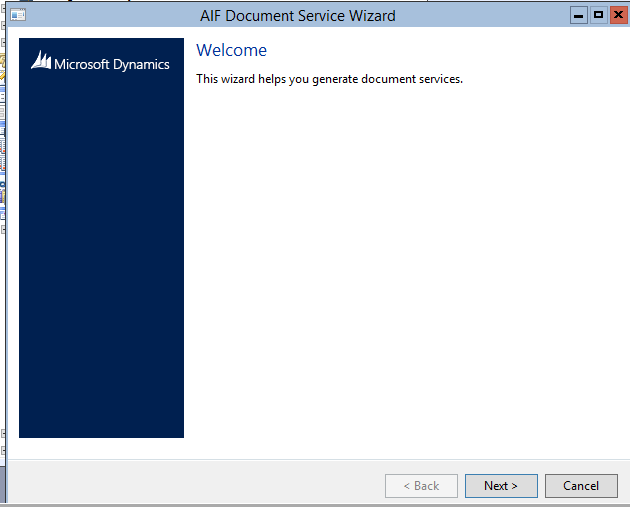


Now we are ready to start to generate the AIF Document Service Wizard.

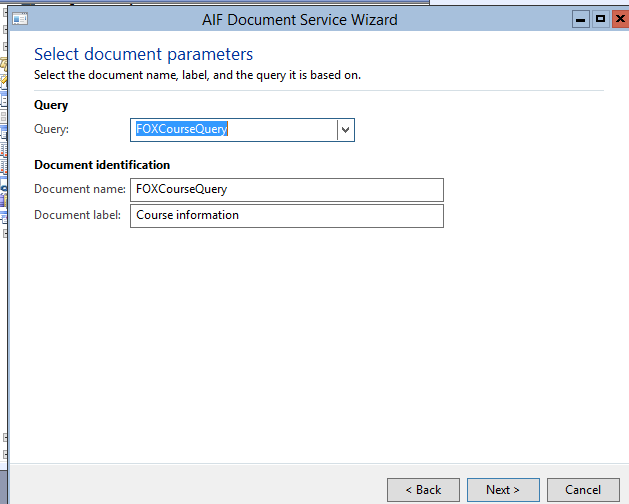
Go to Tools| Wizards| AIF Docuement Service Wizard



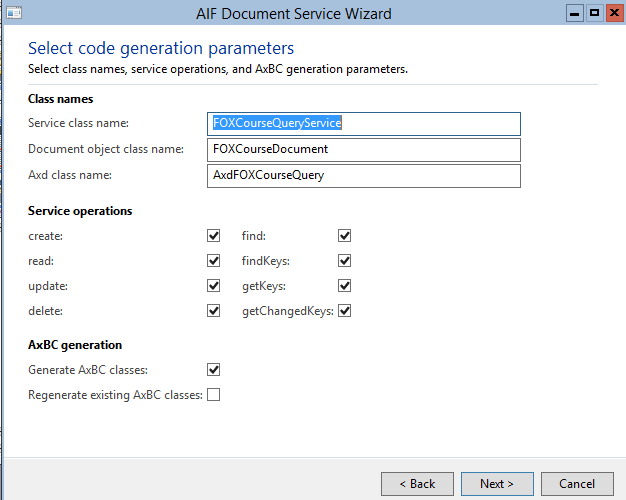
First screen, CLick Next.



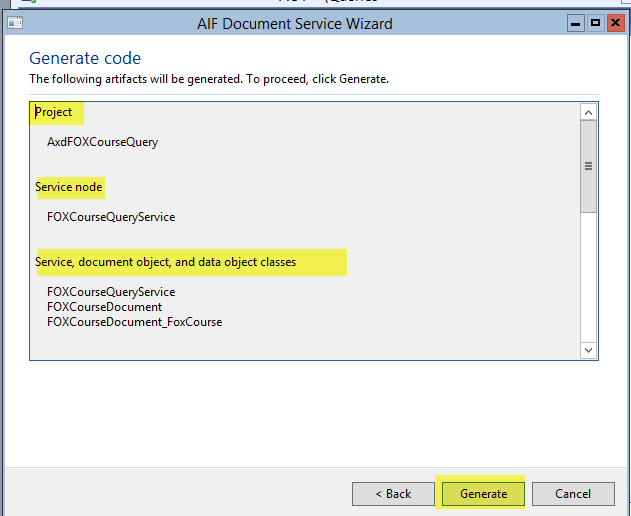
Enter name for the Query and Document.

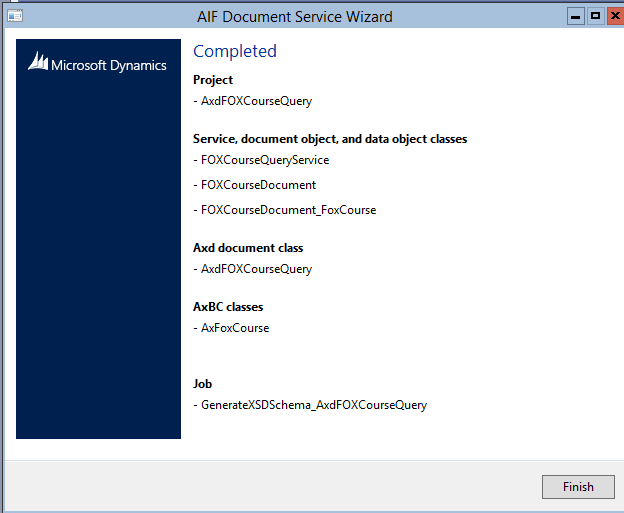


In the next menu enter the name for the Service -, Document- and Axd classes.

Select at least service operators create, read, update, delete and find. I will not describe the findKeys ,getKeys and getChanged Keys in this post. Also important is to select the Generate AxBC classes option. Click 'next'.

In the next screen we see the created objects.





We see that the sytem created :

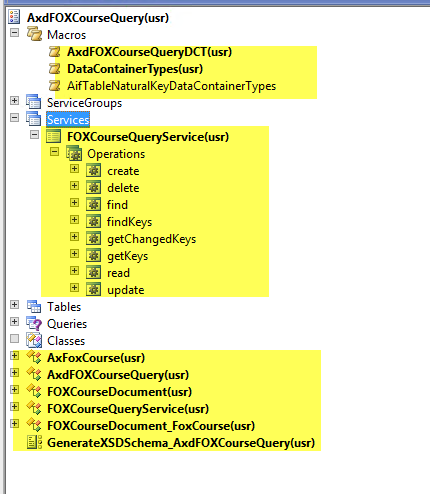
\* document query , contains the table information

\* document class, contains the business logic

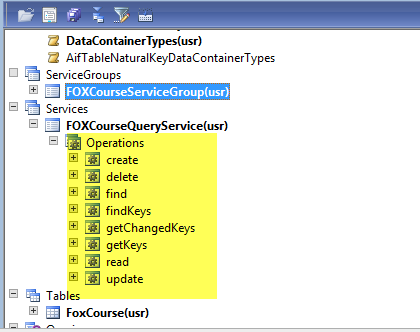
\*document service class for service operations extends the AifDocumentService class

\* AxBc classes we need to perform CRUD operations.

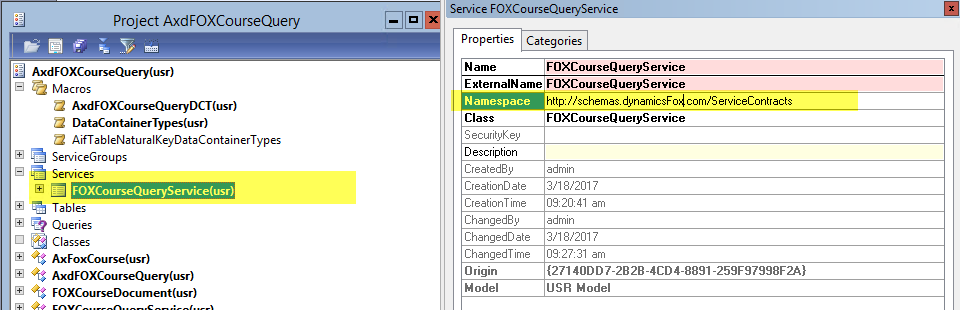
System generate the objects as private project. Here we can see the created objects.



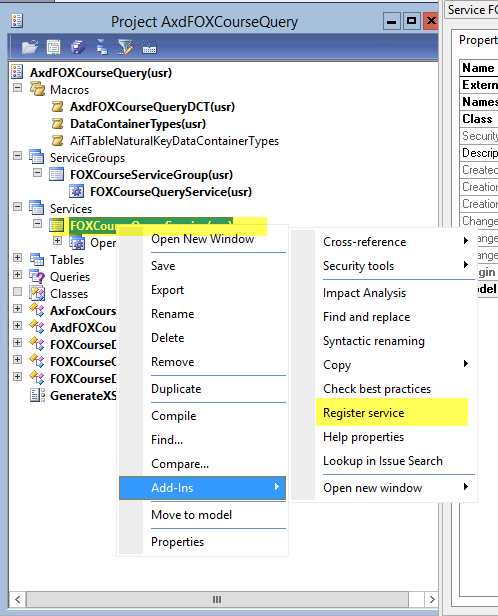
If we open the Service we see the assigend operations.



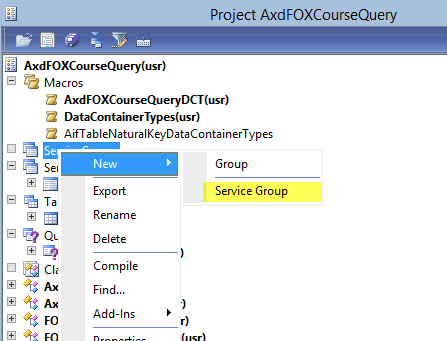
At the service we need to assign the namespace.



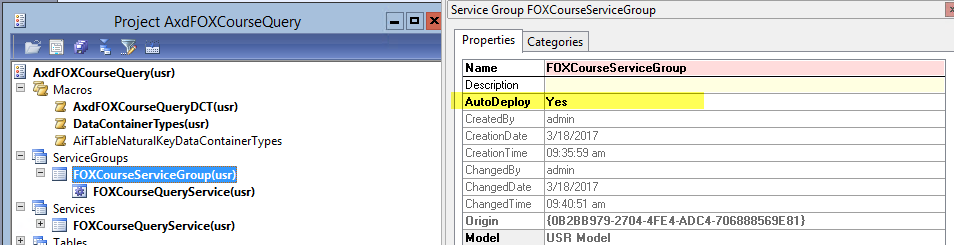
Register the service



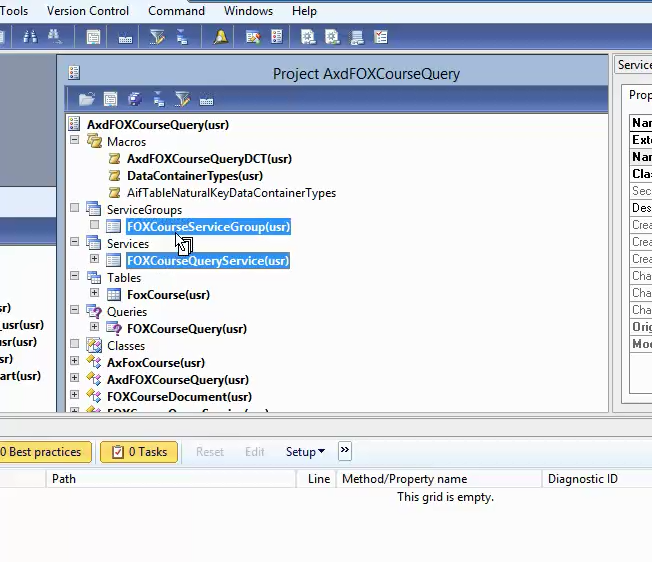
Now we go to create a new service group:



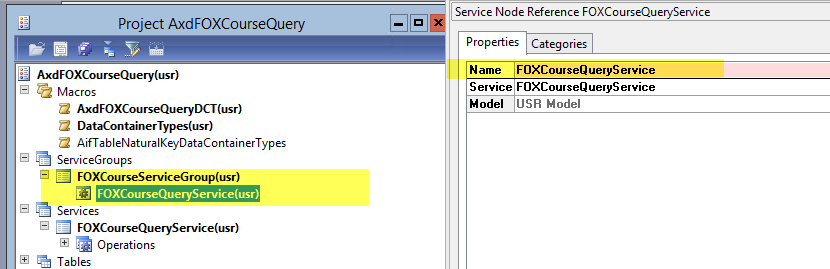
Set in the property menu option 'AutoDeploy' to Yes.



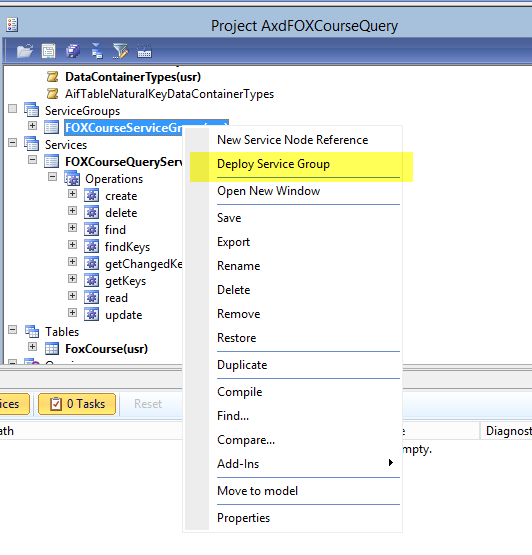
Drag and draw the service to the service group:



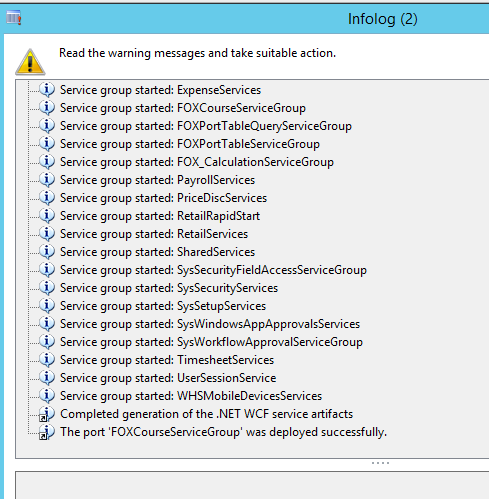
When done, Service group looks like this:



Now we are ready for Deploy the servicegroup. Deploy the service.

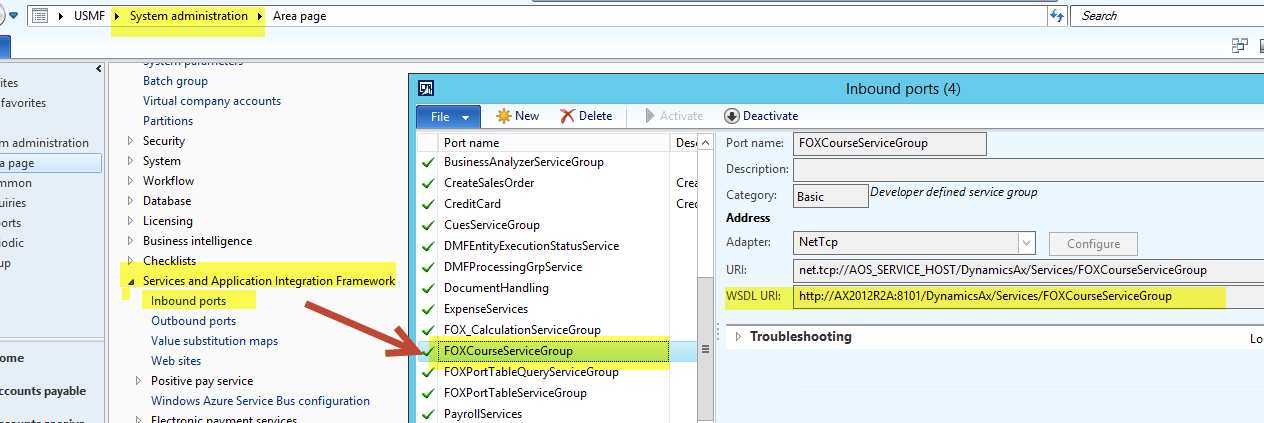


Info log informs us the results of the deployment.

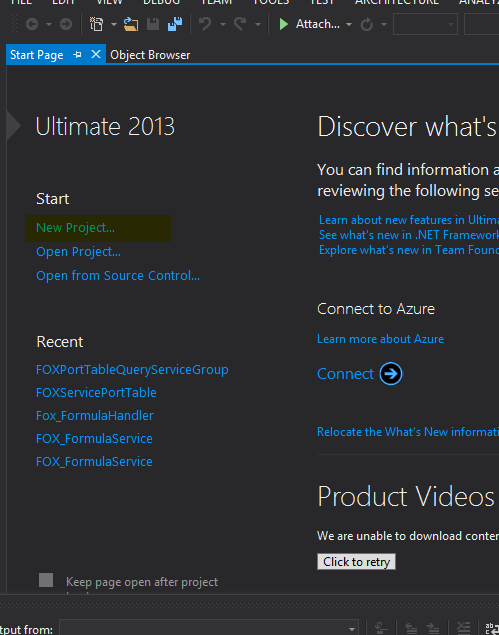


When deployment is ready, the service is available in

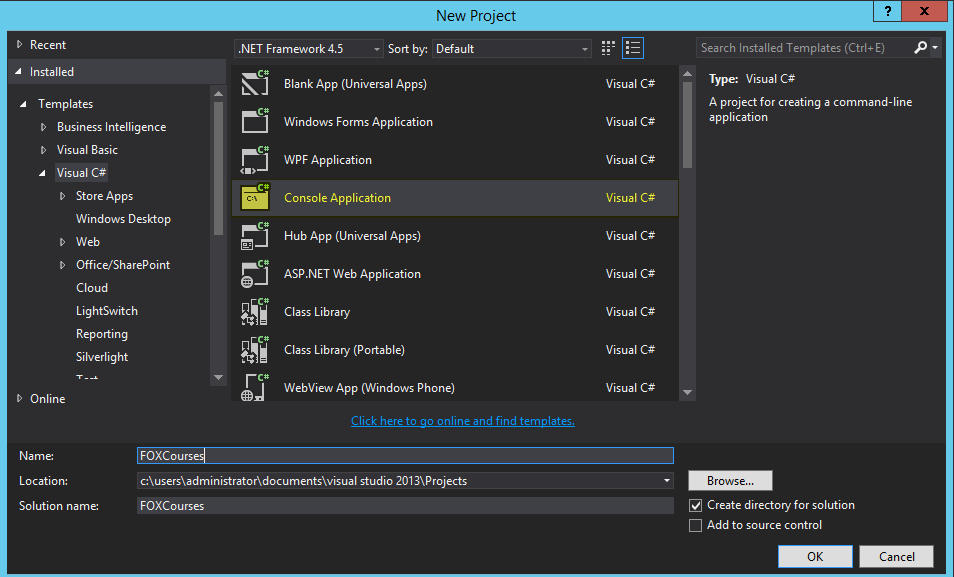
System administration | Services and Application Integration Framework | inbound ports.We need to remeber the WSDL URL as it's necessary in Visual Studio for the Service Reference setup. Service Refence connects Dynamics Ax with Visual Studio, and makes Ax objects availale in Visual Studio.



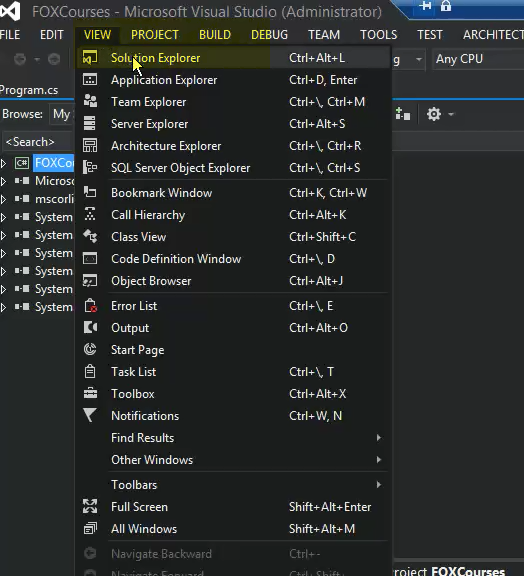
Now open Visual Studio, and open a new project:



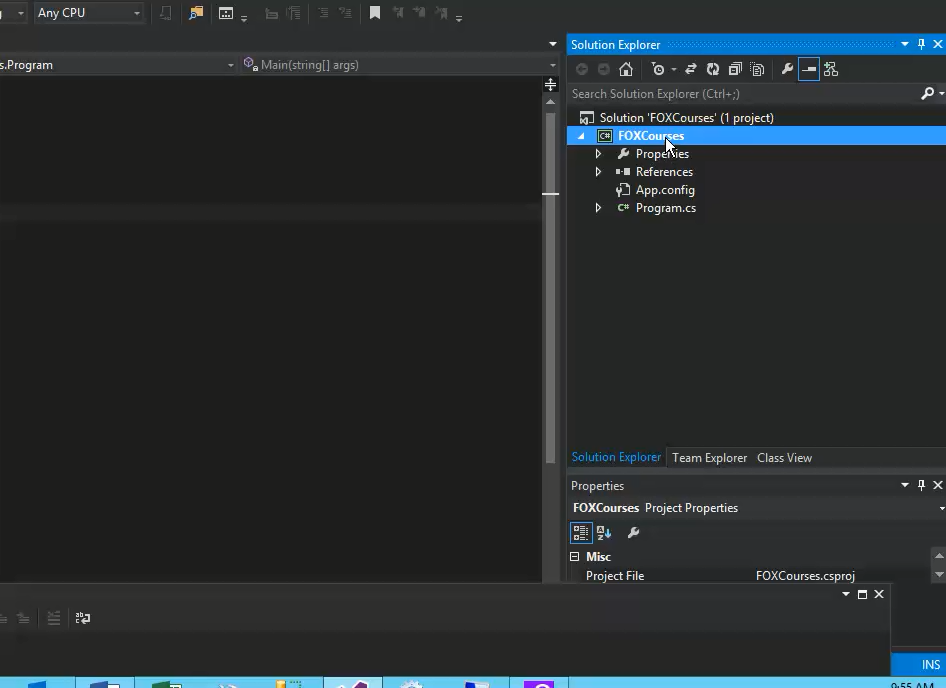
Enter the project name. Choose for Console Application.



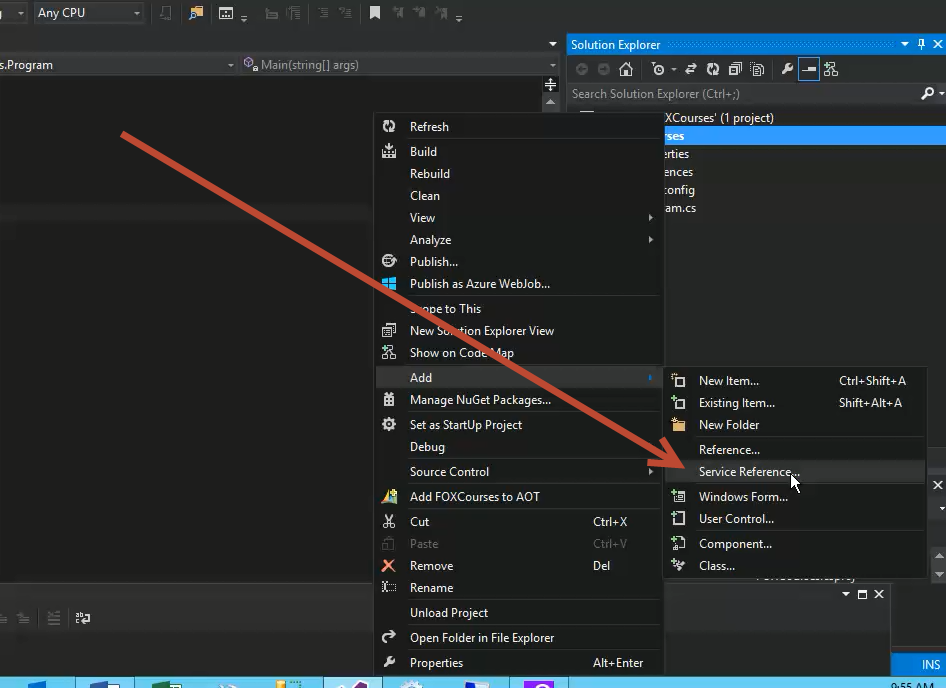
In the next screen be aware that solution explorer screen is available. If not select in the menu bar option 'View', and select solution explorer.



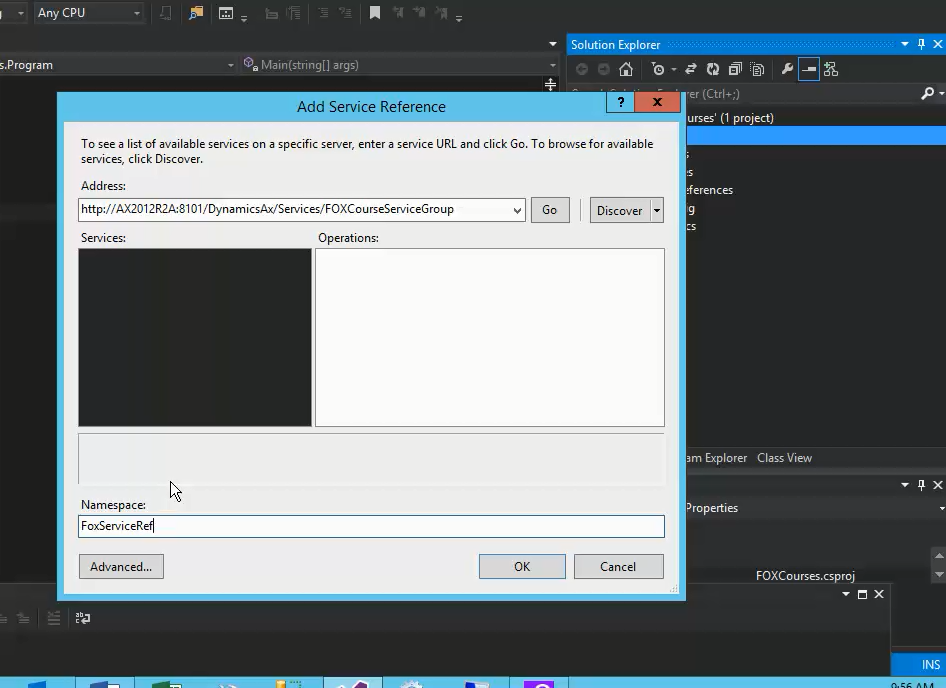
Now put cursor at the project name and do a right mouse click.



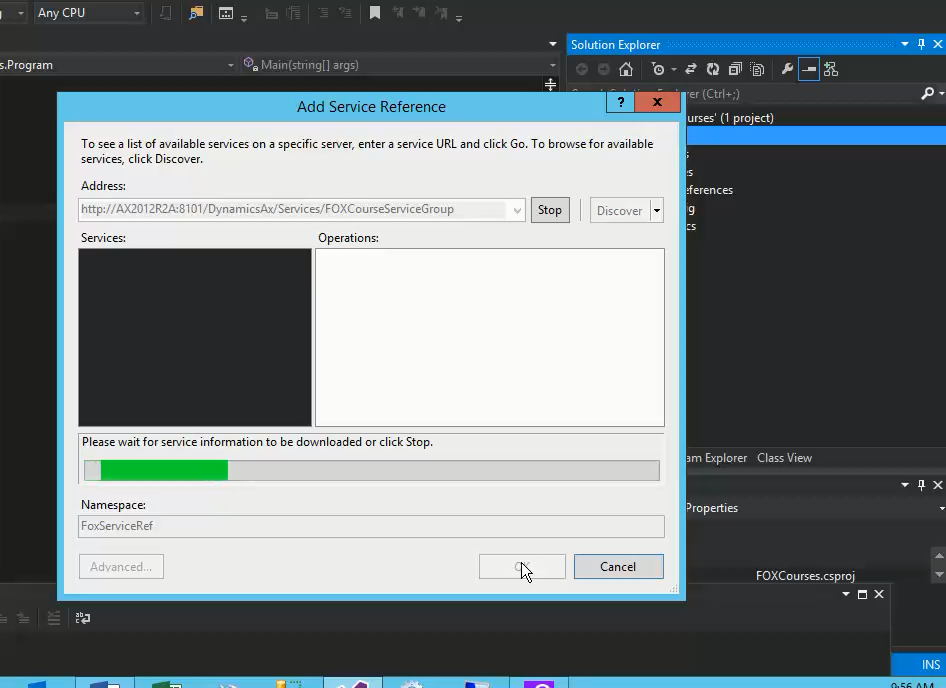
Via option 'add', you can select the Service Reference.



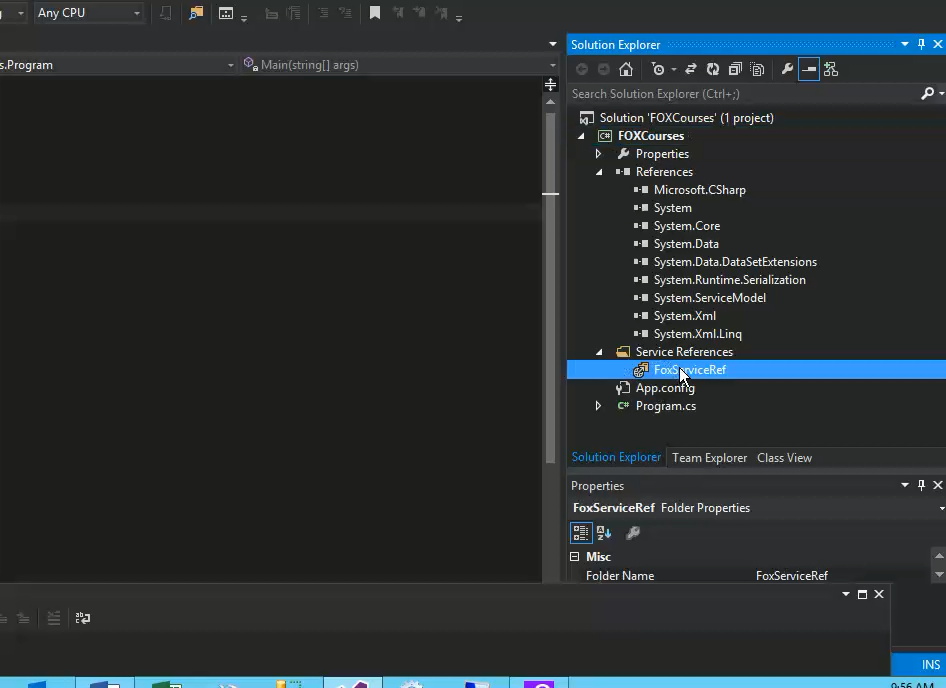
This dialog appers now. Insert in the 'Address' box the WSDL URL from Dynamics Ax.



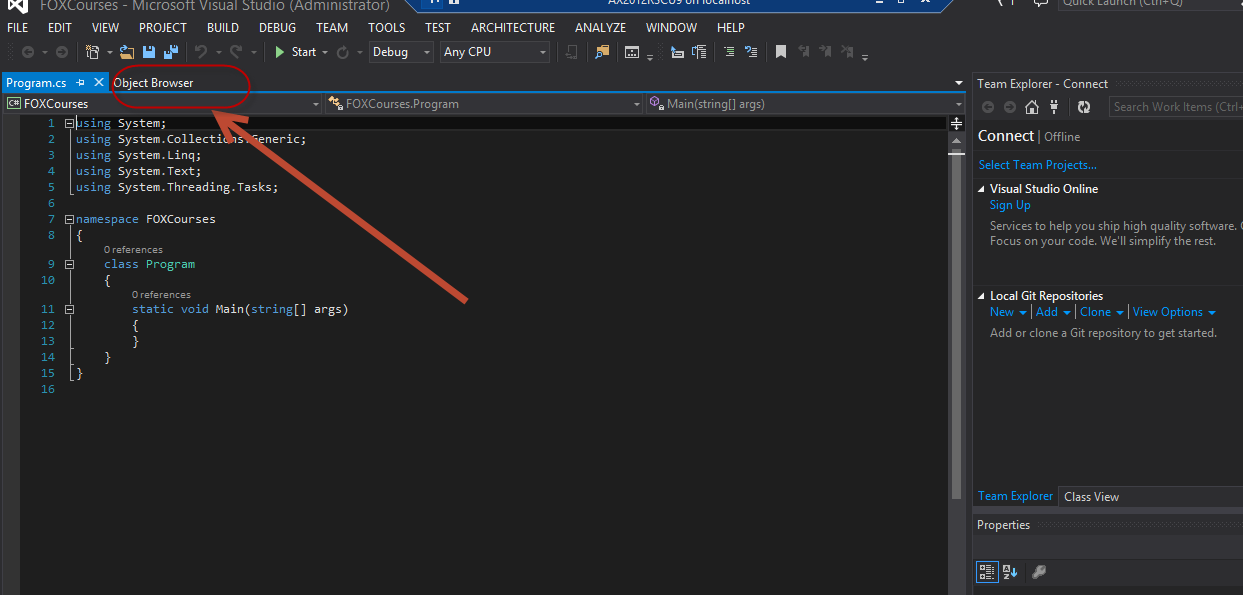
Click OK button.



When done the Solution explorer looks like this:



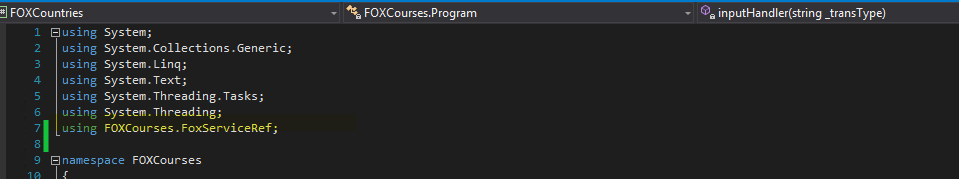
Now we are going to the Program.cs tab. Here we see teh C# code we need to change in order to build the logic for our console.

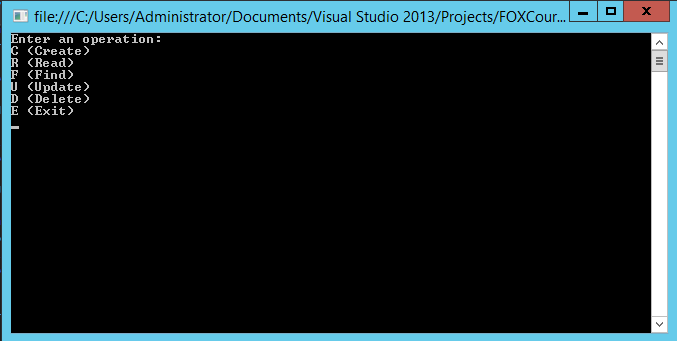


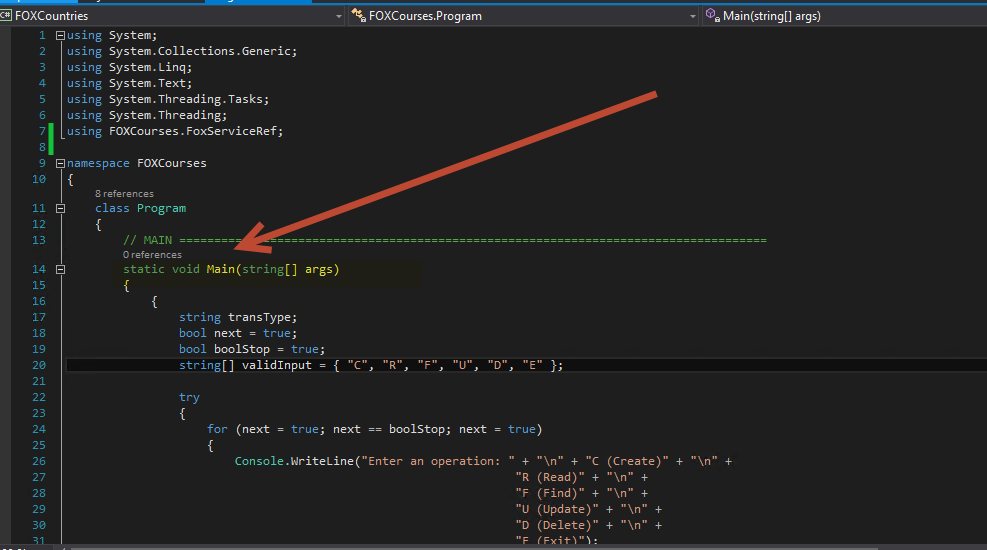
First we make an input menu for the console.

The first modification we are doing is adding the yellow marked line:

so that we can easily instatiate the Ax objects in our program.

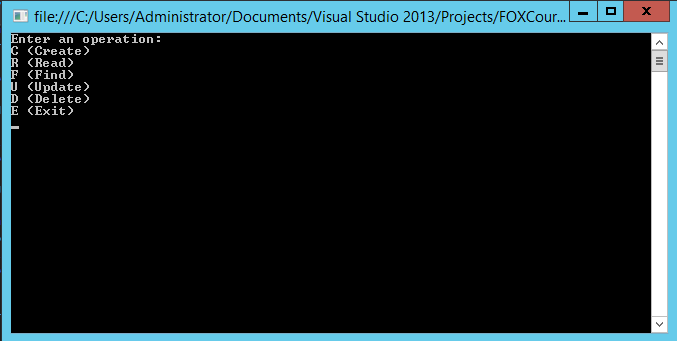


Second step is setting up the 'main' section. To have a Main method in our cs program is mandatory.In our example the logic in the main method presents the console options to the user: 



The fill logic of the main looks like this:

When the console is started this menu will be presented to the user:



The C# logic behind the main is:

static void Main(string[] args)

{

{

string transType;

bool next = true;

bool boolStop = true;

string[] validInput = { "C", "R", "F", "U", "D", "E" };

try

{

for (next = true; next == boolStop; next = true)

{

Console.WriteLine("Enter an operation: " + "\n" + "C (Create)" + "\n" +

"R (Read)" + "\n" +

"F (Find)" + "\n" +

"U (Update)" + "\n" +

"D (Delete)" + "\n" +

"E (Exit)");

transType = (Convert.ToString(Console.ReadLine())).ToUpper();

int pos = Array.IndexOf(validInput, transType);

if (pos < 0)

{

continue;

}

else if (transType == "E")

{

boolStop = false;

}

else

{

Program.selectTransType(transType);

Console.WriteLine("Do you like to continue? (Y/N)");

string request = Console.ReadLine();

boolStop = (request.ToUpper() == "Y") ? true : false;

}

}

}

catch (Exception e)

{

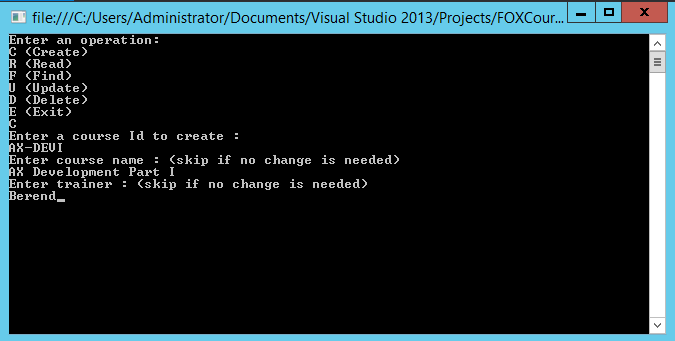
Console.WriteLine("Exception: " + e.Message);

}

}

}

User can make a choice. C# Logic behind you can see in the second next screen dump below, is done in the "Input Handler" and the 'SelectTransType' function the functions are part of the 'Program' section, and we can call it for all functions made in the program.



// INPUT HANDLER ============================================================================================================

static string[] inputHandler(string \_transType)

{

bool printTxt = false; ;

string comment ="";

string courseName = " ";

string trainer = " ";

switch (\_transType)

{

case "C":

comment = "create";

printTxt = true;

break;

case "R":

comment = "read";

break;

case "U":

comment = "update";

printTxt = true;

break;

case "D":

comment = "remove";

break;

}

Console.WriteLine(string.Format("Enter a course Id to {0} : ", comment));

string courseId = Console.ReadLine();

if (printTxt == true)

{

string addText = "(skip if no change is needed)";

Console.WriteLine(string.Format("Enter course name : {0}", addText));

courseName = Console.ReadLine();

Console.WriteLine(string.Format("Enter trainer : {0}", addText));

trainer = Console.ReadLine();

}

string[] container = new string[] { courseId, courseName, trainer, comment };

return container;

}

//SELECTION ===========================================================================================================

static void selectTransType(string transType = "")

{

if (transType == "F")

{

Program.find();

}

else if (transType == "")

{

return;

}

else

{

Program.CRUDOperation(transType);

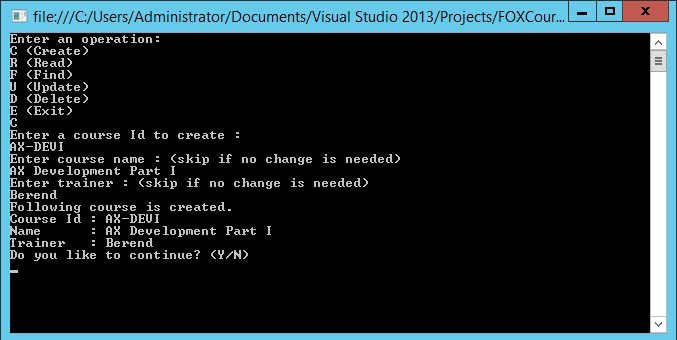
}

return;

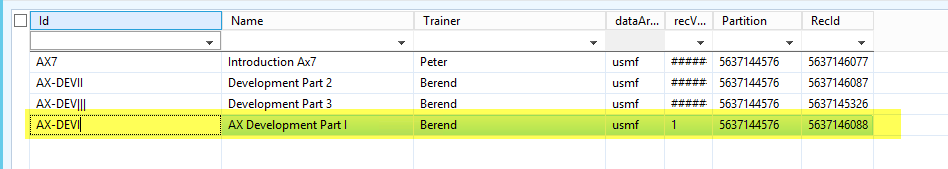
}

CRUD operations in the console. It's possible in the console to create, read, update and delete record in the assigned Ax table.

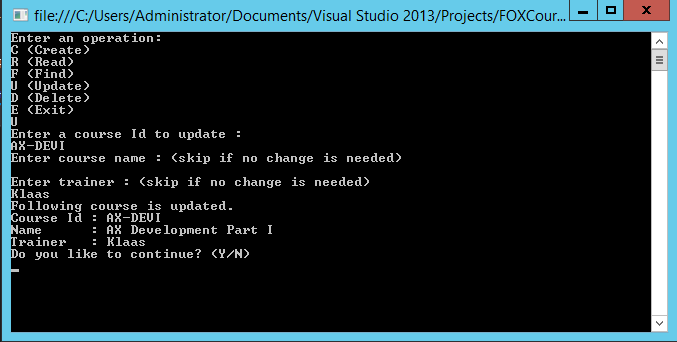
Let;s assume the user is selecting the "C" create option. Then a new record will be created in our Ax table FOXCourse.



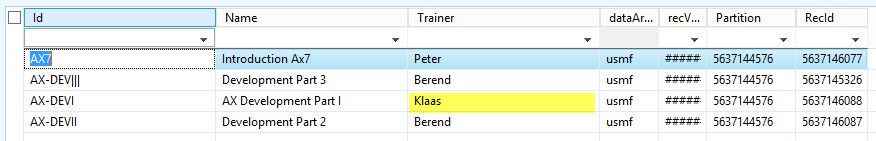
If we check AX we see that the record is created:



We can also update a record:

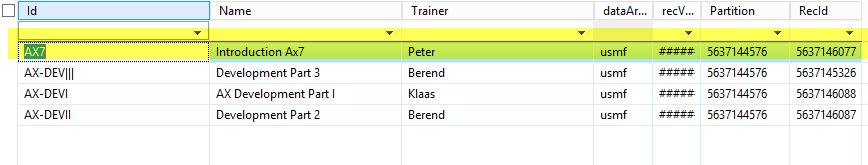


In Ax we see that the selected record is updated:

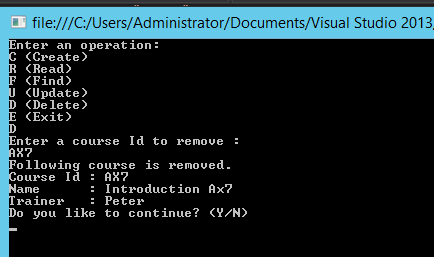


If we enter "D" we can remove a record:

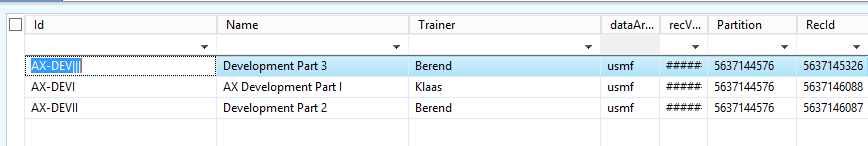
We will choose to delete this record:



We run the console:

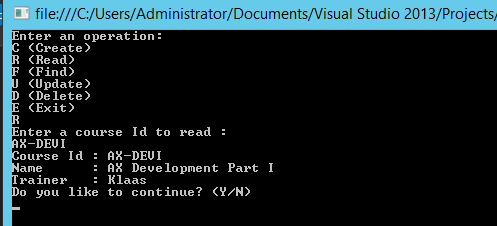


And this is the result:



We see that our selected record is removed.

Another option is the 'Read' option:



The logic behind this functionality is done in the 'CRUDOperation' function and the 'InseryUpdateFields' function.

// INSERT OR UPDATE LOGIC===========================================================================

static void insertUpdateFields(string \_transType, AxdEntity\_FoxCourse \_courseTable, string[] \_array)

{

\_courseTable.Id = (\_transType == "C") ? \_array[0] : \_courseTable.Id;

\_courseTable.Name = \_array[1] == "" ? \_courseTable.Name : \_array[1];

\_courseTable.Trainer = \_array[2] == "" ? \_courseTable.Trainer : \_array[2];

if (\_transType == "U")

{

\_courseTable.action = AxdEnum\_AxdEntityAction.update;

\_courseTable.actionSpecified = true;

}

}

//CRUDOperation========================================================================================

static void CRUDOperation(string \_transType)

{

try

{

AxdFOXCourseQuery courseInfo = null;

AxdEntity\_FoxCourse courseTable = null;

EntityKey entityKey = null; ;

EntityKey[] entityKeys = null;

string[] values = Program.inputHandler(\_transType);

FOXCourseQueryServiceClient client = new FOXCourseQueryServiceClient();

CallContext context = new CallContext();

context.Company = "USMF";

if (\_transType != "C")

{

KeyField keyfield = new KeyField() { Field = "Id", Value = values[0] };

entityKey = new EntityKey();

entityKey.KeyData = new KeyField[1] { keyfield };

entityKeys = new EntityKey[1] { entityKey };

courseInfo = client.read(context, entityKeys);

courseTable = courseInfo.FoxCourse[0];

}

string[] courseData = new string[3] { values[0], values[1], values[2] };

switch (\_transType)

{

case "C" :

courseInfo = new AxdFOXCourseQuery();

courseTable = new AxdEntity\_FoxCourse();

Program.insertUpdateFields(\_transType, courseTable, values);

courseInfo.FoxCourse = new AxdEntity\_FoxCourse[1] { courseTable };

entityKeys = client.create(context, courseInfo);

break;

case "D":

client.delete(context, entityKeys);

break;

case "U":

courseTable = courseInfo.FoxCourse.First();

Program.insertUpdateFields(\_transType, courseTable, courseData);

client.update(context, entityKeys, courseInfo);

break;

}

if (\_transType != "R")

{

Console.WriteLine(String.Format("Following course is {0}d.", values[3]));

}

Console.WriteLine(String.Format("Course Id : {0} " + "\n" +

"Name : {1} " + "\n" +

"Trainer : {2} ",

courseTable.Id,

courseTable.Name,

courseTable.Trainer));

}

catch (Exception e)

{

Console.WriteLine("Exception: " + e.Message);

Console.ReadLine();

}

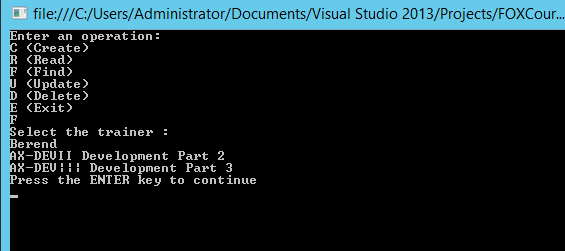
}

Finally we have also configured the 'FInd' operator. The FInd operator is supported by two functions: 'Find' and 'QueryCriteria'. In QueryCriteria we defined the filtering of our selection

.

The functionality will be activated when we select "F" (FInd) option in the console:

In this example we have made a filter (QueryCriteria) on the field 'trainer' . We use the operator 'isEqual' as the input, but there are a lots of more options possible in the QueryCriteria operators.



Here is the logic:

//CRITERIA ===========================================================================================================

static QueryCriteria criteria(string dataSource,

string fieldName,

Operator opr,

string value1,

string value2)

{

QueryCriteria criteria = new QueryCriteria();

CriteriaElement criteriaElement = new CriteriaElement();

criteriaElement.DataSourceName = dataSource;

criteriaElement.FieldName = fieldName;

criteriaElement.Operator = opr;

criteriaElement.Value1 = value1;

criteriaElement.Value2 = value2;

criteria.CriteriaElement = new CriteriaElement[1] { criteriaElement };

return criteria;

}

// FIND ============================================================================================================

static void find()

{

Console.WriteLine("Select the trainer : ");

string trainer = Console.ReadLine();

AxdFOXCourseQuery document = new AxdFOXCourseQuery();

QueryCriteria criteria = Program.criteria("FOXCourse", "Trainer", Operator.Equal, trainer, null);

using (FoxServiceRef.FOXCourseQueryServiceClient client = new

FoxServiceRef.FOXCourseQueryServiceClient())

{

try

{

FoxServiceRef.FOXCourseQueryServiceClient titleDocument = new FoxServiceRef.FOXCourseQueryServiceClient();

document = client.find(null, criteria);

foreach (AxdEntity\_FoxCourse table in document.FoxCourse)

{

Console.WriteLine(table.Id + ' ' + table.Name);

}

Console.WriteLine("Press the ENTER key to continue");

Console.ReadLine();

Program.selectTransType();

}

catch (Exception e)

{

Console.WriteLine("Exception: " + e.Message);

Console.ReadLine();

client.Abort();

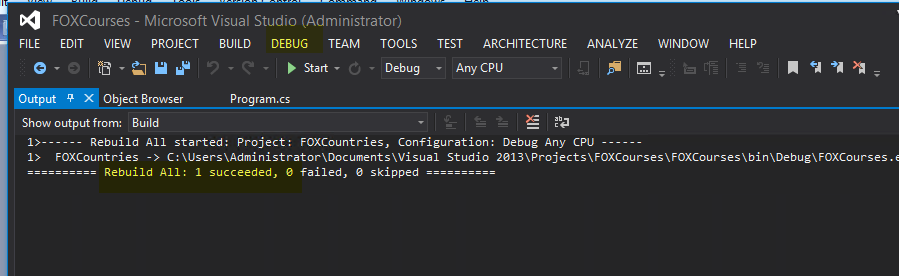
}

}

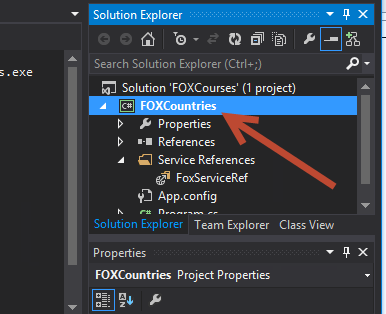
}

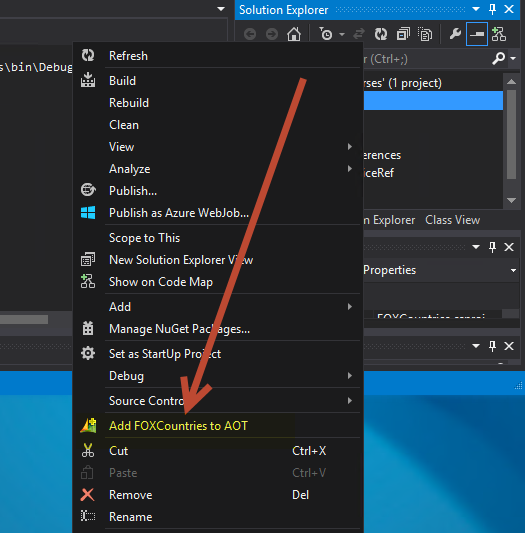
Last step is deployment in Ax:

When rebuilding is succesfully, we can add the project to the AX AOT:

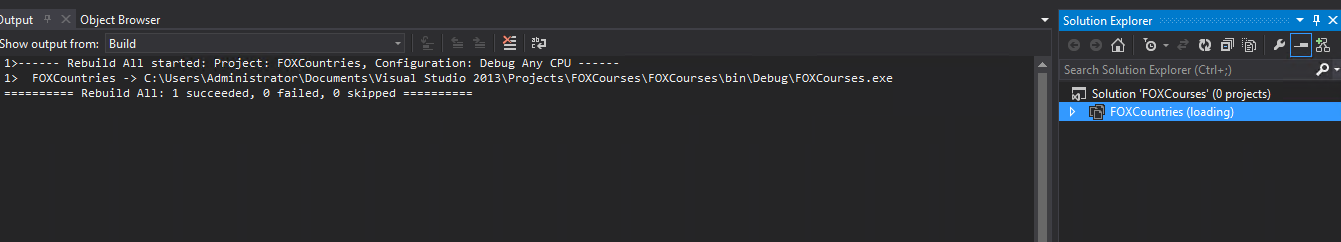


At the poject , in the Solution Explorer screen, you can find the project . Out the cursor on the project and with right mouse click, select option "Add <ProjectName> to AOT





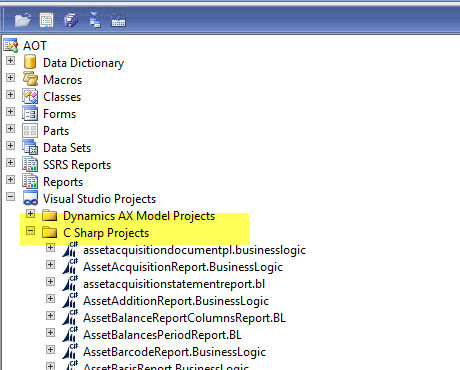
If we see this message then project is succesfully add to AOT:

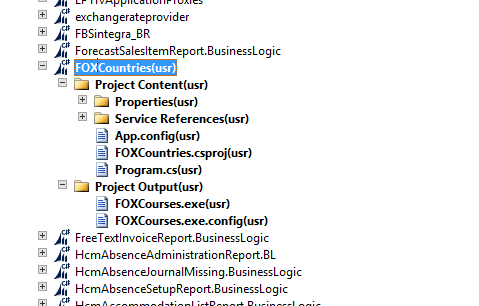


In Solution Explorer section "Program.cs" is added



In AX AOT, under Visual Studio projects, C Sharp projects we will see the project:





APPENDIX: FULL PROGRAM.CS CODE (Visual Studio C#)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

using FOXCourses.FoxServiceRef;

namespace FOXCourses

{

class Program

{

// MAIN ====================================================================================

static void Main(string[] args)

{

{

string transType;

bool next = true;

bool boolStop = true;

string[] validInput = { "C", "R", "F", "U", "D", "E" };

try

{

for (next = true; next == boolStop; next = true)

{

Console.WriteLine("Enter an operation: " + "\n" + "C (Create)" + "\n" +

"R (Read)" + "\n" +

"F (Find)" + "\n" +

"U (Update)" + "\n" +

"D (Delete)" + "\n" +

"E (Exit)");

transType = (Convert.ToString(Console.ReadLine())).ToUpper();

int pos = Array.IndexOf(validInput, transType);

if (pos < 0)

{

continue;

}

else if (transType == "E")

{

boolStop = false;

}

else

{

Program.selectTransType(transType);

Console.WriteLine("Do you like to continue? (Y/N)");

string request = Console.ReadLine();

boolStop = (request.ToUpper() == "Y") ? true : false;

}

}

}

catch (Exception e)

{

Console.WriteLine("Exception: " + e.Message);

}

}

}

//SELECTION ===========================================================================================================

static void selectTransType(string transType = "")

{

if (transType == "F")

{

Program.find();

}

else if (transType == "")

{

return;

}

else

{

Program.CRUDOperation(transType);

}

return;

}

//CRITERIA ===========================================================================================================

static QueryCriteria criteria(string dataSource,

string fieldName,

Operator opr,

string value1,

string value2)

{

QueryCriteria criteria = new QueryCriteria();

CriteriaElement criteriaElement = new CriteriaElement();

criteriaElement.DataSourceName = dataSource;

criteriaElement.FieldName = fieldName;

criteriaElement.Operator = opr;

criteriaElement.Value1 = value1;

criteriaElement.Value2 = value2;

criteria.CriteriaElement = new CriteriaElement[1] { criteriaElement };

return criteria;

}

// FIND ============================================================================================================

static void find()

{

Console.WriteLine("Select the trainer : ");

string trainer = Console.ReadLine();

AxdFOXCourseQuery document = new AxdFOXCourseQuery();

QueryCriteria criteria = Program.criteria("FOXCourse", "Trainer", Operator.Equal, trainer, null);

using (FoxServiceRef.FOXCourseQueryServiceClient client = new

FoxServiceRef.FOXCourseQueryServiceClient())

{

try

{

FoxServiceRef.FOXCourseQueryServiceClient titleDocument = new FoxServiceRef.FOXCourseQueryServiceClient();

document = client.find(null, criteria);

foreach (AxdEntity\_FoxCourse table in document.FoxCourse)

{

Console.WriteLine(table.Id + ' ' + table.Name);

}

Console.WriteLine("Press the ENTER key to continue");

Console.ReadLine();

Program.selectTransType();

}

catch (Exception e)

{

Console.WriteLine("Exception: " + e.Message);

Console.ReadLine();

client.Abort();

}

}

}

// INPUT HANDLER ============================================================================================================

static string[] inputHandler(string \_transType)

{

bool printTxt = false; ;

string comment ="";

string courseName = " ";

string trainer = " ";

switch (\_transType)

{

case "C":

comment = "create";

printTxt = true;

break;

case "R":

comment = "read";

break;

case "U":

comment = "update";

printTxt = true;

break;

case "D":

comment = "remove";

break;

}

Console.WriteLine(string.Format("Enter a course Id to {0} : ", comment));

string courseId = Console.ReadLine();

if (printTxt == true)

{

string addText = "(skip if no change is needed)";

Console.WriteLine(string.Format("Enter course name : {0}", addText));

courseName = Console.ReadLine();

Console.WriteLine(string.Format("Enter trainer : {0}", addText));

trainer = Console.ReadLine();

}

string[] container = new string[] { courseId, courseName, trainer, comment };

return container;

}

// INSERT OR UPDATE LOGIC===========================================================================

static void insertUpdateFields(string \_transType, AxdEntity\_FoxCourse \_courseTable, string[] \_array)

{

\_courseTable.Id = (\_transType == "C") ? \_array[0] : \_courseTable.Id;

\_courseTable.Name = \_array[1] == "" ? \_courseTable.Name : \_array[1];

\_courseTable.Trainer = \_array[2] == "" ? \_courseTable.Trainer : \_array[2];

if (\_transType == "U")

{

\_courseTable.action = AxdEnum\_AxdEntityAction.update;

\_courseTable.actionSpecified = true;

}

}

//CRUDOperation========================================================================================

static void CRUDOperation(string \_transType)

{

try

{

AxdFOXCourseQuery courseInfo = null;

AxdEntity\_FoxCourse courseTable = null;

EntityKey entityKey = null; ;

EntityKey[] entityKeys = null;

string[] values = Program.inputHandler(\_transType);

FOXCourseQueryServiceClient client = new FOXCourseQueryServiceClient();

CallContext context = new CallContext();

context.Company = "USMF";

if (\_transType != "C")

{

KeyField keyfield = new KeyField() { Field = "Id", Value = values[0] };

entityKey = new EntityKey();

entityKey.KeyData = new KeyField[1] { keyfield };

entityKeys = new EntityKey[1] { entityKey };

courseInfo = client.read(context, entityKeys);

courseTable = courseInfo.FoxCourse[0];

}

string[] courseData = new string[3] { values[0], values[1], values[2] };

switch (\_transType)

{

case "C" :

courseInfo = new AxdFOXCourseQuery();

courseTable = new AxdEntity\_FoxCourse();

Program.insertUpdateFields(\_transType, courseTable, values);

courseInfo.FoxCourse = new AxdEntity\_FoxCourse[1] { courseTable };

entityKeys = client.create(context, courseInfo);

break;

case "D":

client.delete(context, entityKeys);

break;

case "U":

courseTable = courseInfo.FoxCourse.First();

Program.insertUpdateFields(\_transType, courseTable, courseData);

client.update(context, entityKeys, courseInfo);

break;

}

if (\_transType != "R")

{

Console.WriteLine(String.Format("Following course is {0}d.", values[3]));

}

Console.WriteLine(String.Format("Course Id : {0} " + "\n" +

"Name : {1} " + "\n" +

"Trainer : {2} ",

courseTable.Id,

courseTable.Name,

courseTable.Trainer));

}

catch (Exception e)

{

Console.WriteLine("Exception: " + e.Message);

Console.ReadLine();

}

}

}

}