OpenEdge Data Provider

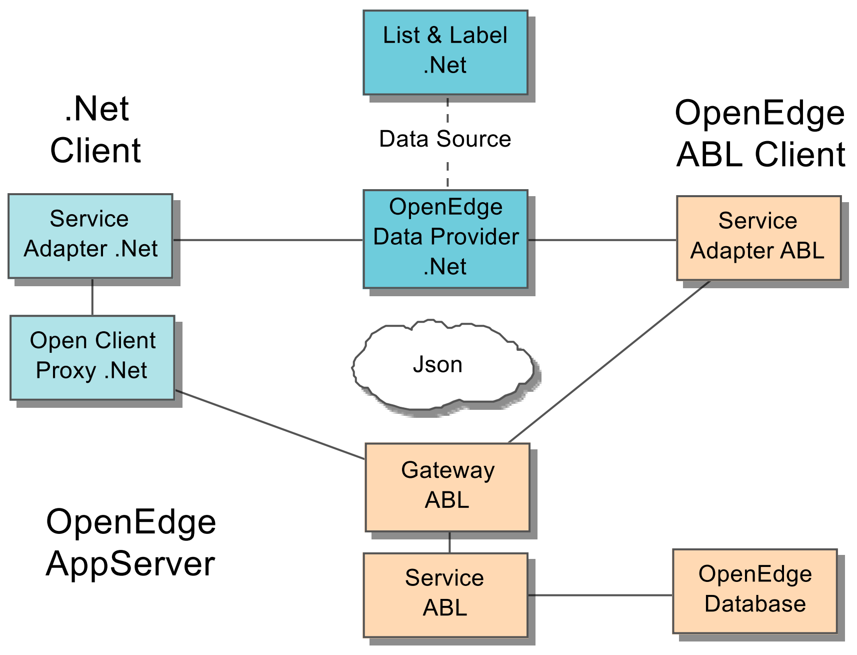
Created by Thomas Wurl, Taste IT Consulting

combit List & Label

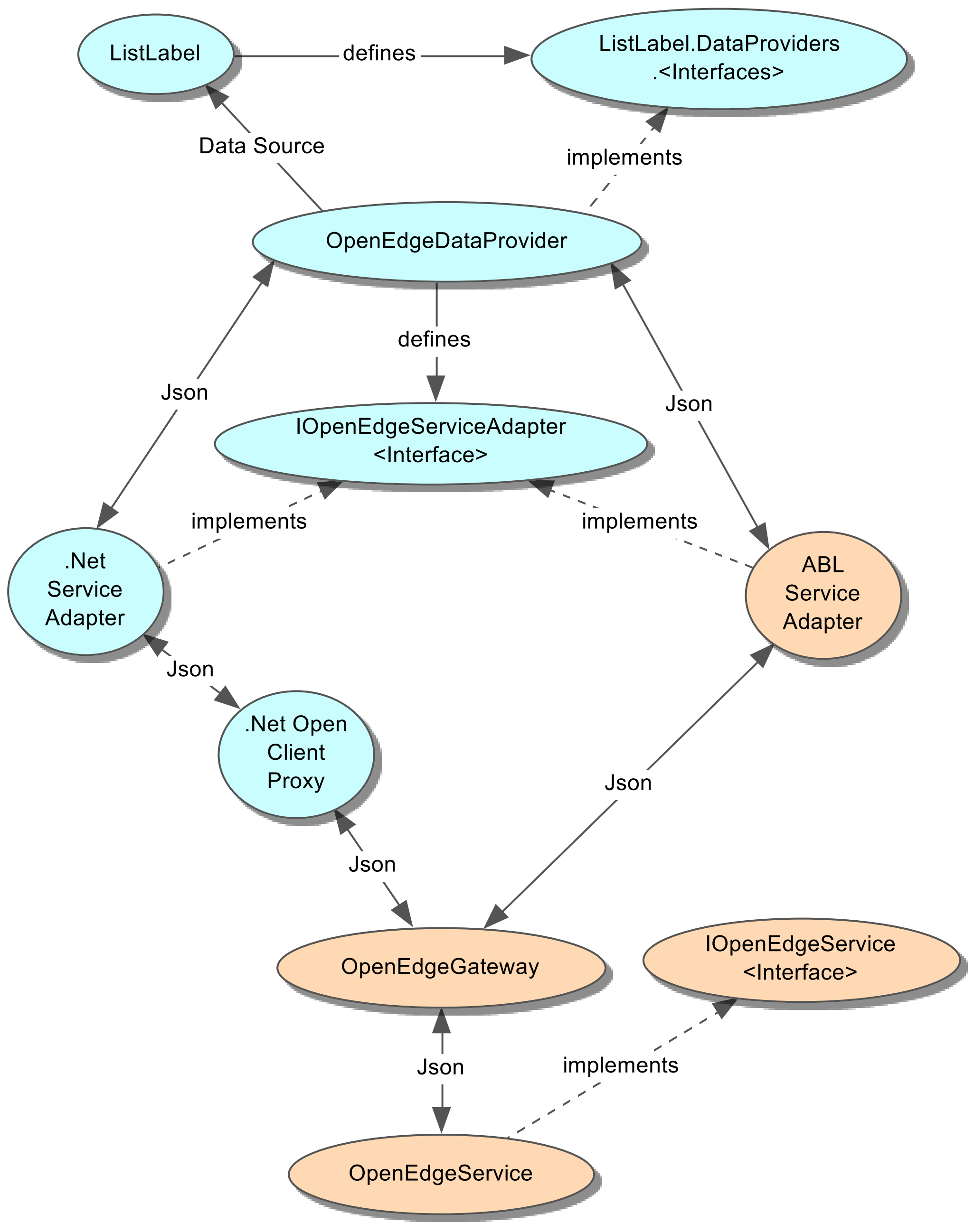
Content

Introduction

The OpenEdge data provider for List & Label is written in .NET C#. The communciation between List & Label and OpenEdge is done via JSON.



Architecture



List & Label defines some interfaces that need to be implemented by a data provider. A data provider is then bound to List & Label as a data source. What data List & Label asks for is driven by the layout and a provider doesn't even know about List & Label.

The interface IServiceAdapter is defined in the data provider and needs to be implemented either in .NET (.NET Application) or in ABL (ABL Application). Being able to use .NET on the client side also enables the use of combit Report Server or List & Label web reporting based on ASP.

The service adapter interface is quite simple and requires only two methods:

```
public interface IServiceAdapter
{
    bool GetSchema (string ServiceName, OELongchar JsonServiceParameter, out OELongchar JsonSchema);
    bool GetData   (string ServiceName, OELongchar JsonServiceParameter, OELongchar JsonDataRequest, out OELongchar JsonDataResponse);
}

// Container for a json string passed as parameter to and from OpenEdge.
public class OELongchar
{
    public string Data { get; set; }
}
```

Since OpenEdge has problems to map .NET strings to LONGCHAR, JSON data is transferred in an OELongchar container class.

A service is a class in OpenEdge that is able to describe its own schema and to send data based on a request. By default, the class name is used as service name.

The gateway creates an instance of a service with dynamic-new (<service name>) and talks to the service via the interface IOpenEdgeService.

```
INTERFACE ListLabel.OpenEdgeAdapter.IOpenEdgeService:

    METHOD PUBLIC LONGCHAR getSchema( plcServiceParameterJson AS LONGCHAR ).
    METHOD PUBLIC LONGCHAR getData   ( plcServiceParameterJson AS LONGCHAR, plcRequestJson AS LONGCHAR ).

END INTERFACE.
```

The data provider comes with a generic and very powerful service class. You just register your entire database and you are done. The service also works for a memory dataset either on the client or app server.

It can also be used to access all kind of your own classes like Business Entities. But then you have to decide how you want to filter the data in your classes. You may have to translate the JSON request to something your classes understand.

To send parameters to your own services you can also register service parameters in the data provider that will be passed to your service. In this scenario you probably have some forms on your client to enter your parameters or define your parameters in the context of your application.

As soon as you have the filtered data inside your class, you just pass the handle of your dataset and the generic service takes care of it to extract only the values List & Label asks for and to do the required sorting.

OpenEdge Services

Database services

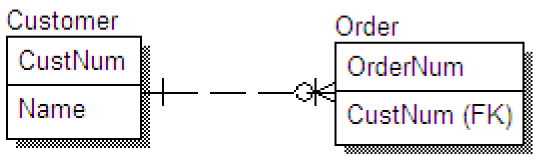
A database service is a service that gives List & Label access to an entire database. A database service always runs with a direct database connection (shared memory connection recommended), but can be accessed through the app server.

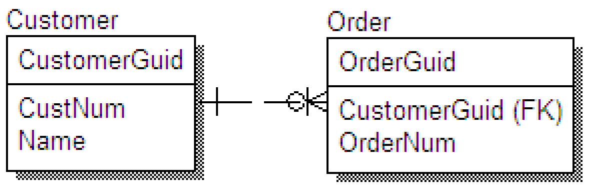
The task that you have is to register all tables of your database and all relations between these tables. The tables are easily taken from the meta schema.

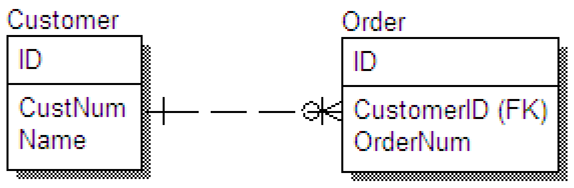
Unfortunately, the OpenEdge database doesn't know about relations. If you already have information about your relations somewhere you can use that information to generate an include file for your service, e.g. "sports2000_relations.i" in the "ListLabelDemo" folder. Another possibility is to use the "dbschema" tool in the "schema" subfolder. By that, all possible relations in a database can be retrieved, the following rules apply:

- A parent table inherits all fields of a unique index into a child table.
- The fields must match in name and data type.
- The fields build a foreign key in the child table and should be leading components of an index of the child table (for performance reason). If the foreign keys build a unique index it's a one-one relations otherwise a one-many relation.

The tool may find too much, especially in databases with cascading keys. But every database is different and may have different naming conventions that only you know. The tool supports designs where "OF" works.





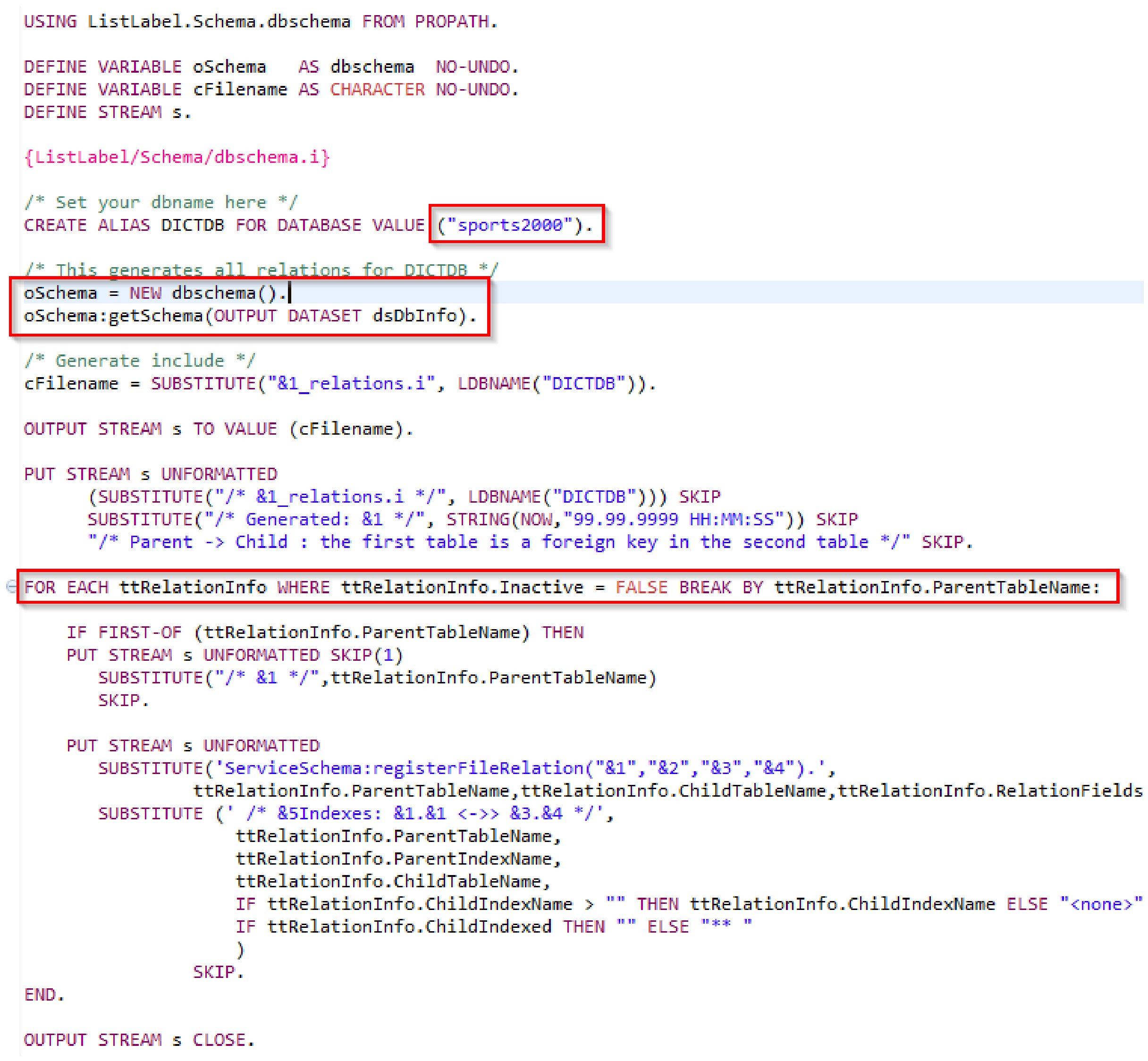


The first two relations will be found by the tool. The third will require some changes in the tool. Unfortunately, the third design is often even mixed with the other two versions in a single database!

Generating database relations

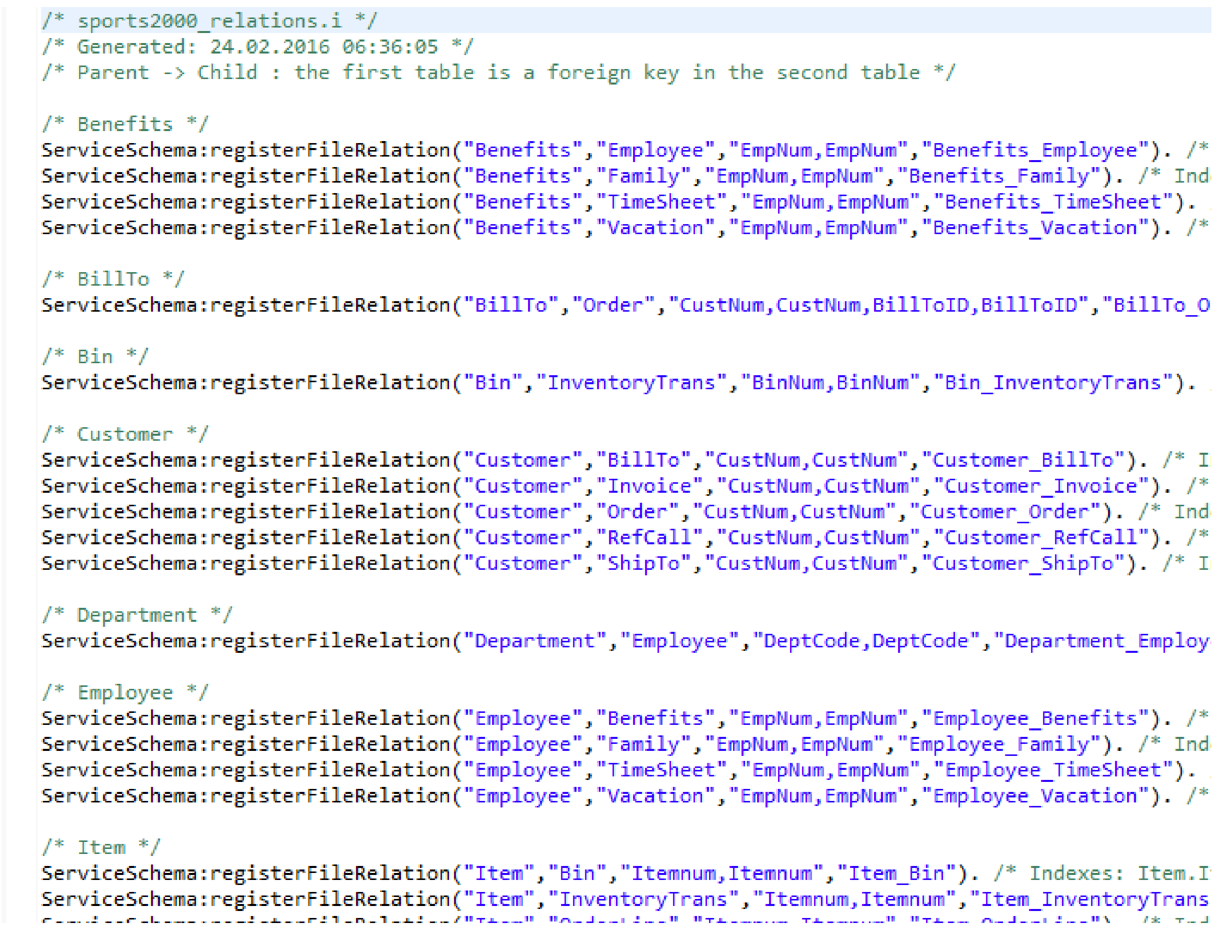
Let's have a look at the sports 2000 database. Its design is version 1.

Run the tool ListLabel/Schema/GenerateDbRelations.p. For your own database change the alias definition.



```
USING ListLabel.Schema.dbschema FROM PROPATH.

DEFINE VARIABLE oSchema   AS dbschema  NO-UNDO.
DEFINE VARIABLE cFilename AS CHARACTER NO-UNDO.
DEFINE STREAM s.

{ListLabel/Schema/dbschema.i}

/* Set your dbname here */
CREATE ALIAS DICTDB FOR DATABASE VALUE ("sports2000").

/* This generates all relations for DICTDB */
oSchema = NEW dbschema().
oSchema:getSchema(OUTPUT DATASET dsDbInfo).

/* Generate include */
cFilename = SUBSTITUTE("&1_relations.i", LDBNAME("DICTDB")).

OUTPUT STREAM s TO VALUE (cFilename).

PUT STREAM s UNFORMATTED
      (SUBSTITUTE("/* &1_relations.i */", LDBNAME("DICTDB"))) SKIP
      SUBSTITUTE("/* Generated: &1 */", STRING(NOW,"99.99.9999 HH:MM:SS")) SKIP
      "/* Parent -> Child : the first table is a foreign key in the second table */" SKIP.

FOR EACH ttRelationInfo WHERE ttRelationInfo.Inactive = FALSE BREAK BY ttRelationInfo.ParentTableName:

    IF FIRST-OF (ttRelationInfo.ParentTableName) THEN
    PUT STREAM s UNFORMATTED SKIP(1)
       SUBSTITUTE("/* &1 */",ttRelationInfo.ParentTableName)
       SKIP.

    PUT STREAM s UNFORMATTED
       SUBSTITUTE('ServiceSchema:registerFileRelation("&1","&2","&3","&4").',
                 ttRelationInfo.ParentTableName,ttRelationInfo.ChildTableName,ttRelationInfo.RelationFields
       SUBSTITUTE (' /* &5Indexes: &1.&1 <->> &3.&4 */',
                     ttRelationInfo.ParentTableName,
                     ttRelationInfo.ParentIndexName,
                     ttRelationInfo.ChildTableName,
                     IF ttRelationInfo.ChildIndexName > "" THEN ttRelationInfo.ChildIndexName ELSE "<none>"
                     IF ttRelationInfo.ChildIndexed THEN "" ELSE "** "
                     )
                 SKIP.
END.

OUTPUT STREAM s CLOSE.
```

The result is in <dbname>_relations.i. For sports 2000 it looks like this:



```
/* sports2000_relations.i */
/* Generated: 24.02.2016 06:36:05 */
/* Parent -> Child : the first table is a foreign key in the second table */

/* Benefits */
ServiceSchema:registerFileRelation("Benefits","Employee","EmpNum,EmpNum","Benefits_Employee"). /*
ServiceSchema:registerFileRelation("Benefits","Family","EmpNum,EmpNum","Benefits_Family"). /* Ind
ServiceSchema:registerFileRelation("Benefits","TimeSheet","EmpNum,EmpNum","Benefits_TimeSheet").
ServiceSchema:registerFileRelation("Benefits","Vacation","EmpNum,EmpNum","Benefits_Vacation"). /*

/* BillTo */
ServiceSchema:registerFileRelation("BillTo","Order","CustNum,CustNum,BillToID,BillToID","BillTo_O

/* Bin */
ServiceSchema:registerFileRelation("Bin","InventoryTrans","BinNum,BinNum","Bin_InventoryTrans").

/* Customer */
ServiceSchema:registerFileRelation("Customer","BillTo","CustNum,CustNum","Customer_BillTo"). /* I
ServiceSchema:registerFileRelation("Customer","Invoice","CustNum,CustNum","Customer_Invoice"). /*
ServiceSchema:registerFileRelation("Customer","Order","CustNum,CustNum","Customer_Order"). /* Ind
ServiceSchema:registerFileRelation("Customer","RefCall","CustNum,CustNum","Customer_RefCall"). /*
ServiceSchema:registerFileRelation("Customer","ShipTo","CustNum,CustNum","Customer_ShipTo"). /* I

/* Department */
ServiceSchema:registerFileRelation("Department","Employee","DeptCode,DeptCode","Department_Employ

/* Employee */
ServiceSchema:registerFileRelation("Employee","Benefits","EmpNum,EmpNum","Employee_Benefits"). /*
ServiceSchema:registerFileRelation("Employee","Family","EmpNum,EmpNum","Employee_Family"). /* Ind
ServiceSchema:registerFileRelation("Employee","TimeSheet","EmpNum,EmpNum","Employee_TimeSheet").
ServiceSchema:registerFileRelation("Employee","Vacation","EmpNum,EmpNum","Employee_Vacation"). /*

/* Item */
ServiceSchema:registerFileRelation("Item","Bin","Itemnum,Itemnum","Item_Bin"). /* Indexes: Item.I
ServiceSchema:registerFileRelation("Item","InventoryTrans","Itemnum,Itemnum","Item_InventoryTrans
ServiceSchema:registerFileRelation("Item","OrderLine","Itemnum,Itemnum","Item_OrderLine"). /* Ind
```

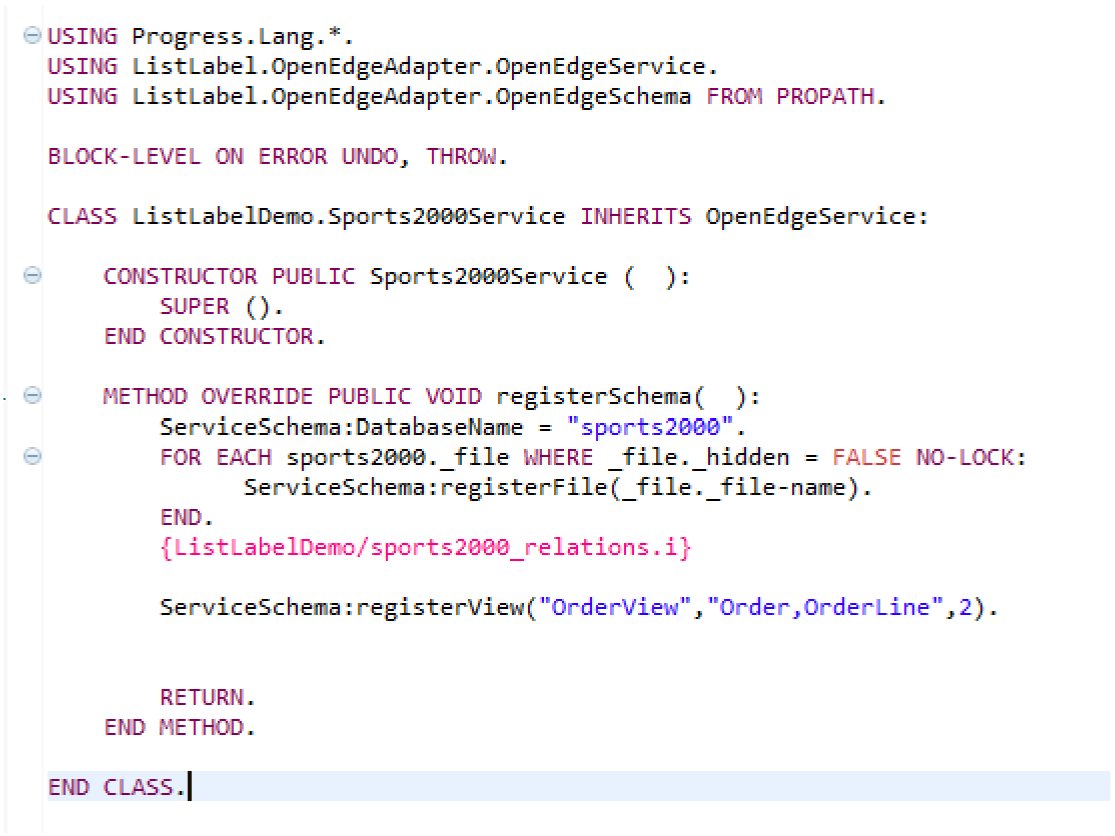
For the sports 2000 database all relations seem to be valid. With your own database you may have to remove or add relations manually. Relations where a parent table plays different roles (like billing address, shipping address) will not be found by the tool because the field names don't match here.

You may also want to filter relations where the child table is properly indexed, meaning that the relation fields are leading components of an index. Do this by filtering:

FOR EACH ttRelationInfo WHERE ChildIndexed = TRUE:

Creating a service class

Open ListLabelDemo/Sports2000Service.cls



```
USING Progress.Lang.*.
USING ListLabel.OpenEdgeAdapter.OpenEdgeService.
USING ListLabel.OpenEdgeAdapter.OpenEdgeSchema FROM PROPATH.

BLOCK-LEVEL ON ERROR UNDO, THROW.

CLASS ListLabelDemo.Sports2000Service INHERITS OpenEdgeService:

    CONSTRUCTOR PUBLIC Sports2000Service (  ):
        SUPER ().
    END CONSTRUCTOR.

    METHOD OVERRIDE PUBLIC VOID registerSchema(  ):
        ServiceSchema:DatabaseName = "sports2000".
        FOR EACH sports2000._file WHERE _file._hidden = FALSE NO-LOCK:
              ServiceSchema:registerFile(_file._file-name).
        END.
        {ListLabelDemo/sports2000_relations.i}

        ServiceSchema:registerView("OrderView","Order,OrderLine",2).


        RETURN.
    END METHOD.

END CLASS.
```

A service inherits from the abstract class OpenEdgeService and must implement the registerSchema() method. The property ServiceSchema is of type ListLabel.OpenEdgeAdapter.OpenEdgeSchema, holds the definitions and has the method getSchemaJson().

For you own Service create a copy of this class in a different namespace, change names and use your own generated include. That's all.

If your database has a big schema and many relations, you will recognize an overhead when the List & Label designer starts. The more objects have to be registered, the longer it takes. This might first be encountered with schemas with more than 100 tables. That's the reason why the following construct of views was added to the schema.

```
ServiceSchema:registerView("OrderView","Order,OrderLine",2).
```

The first parameter is the name of the view. The second is a table list for the base tables you want to have in the view. The third parameter is the foreign key depth for foreign key to be added to the list of tables. Foreign keys and relations are added automatically.
You can also put all the tables you need inside the list and set the foreign key depth = 0.

Calculated columns

Calculated columns are a feature for a database service. This allows you to add additional columns to a database schema table and to fill them whenever needed.

There is a method registerCalculatedColumn in the service schema. Before adding a column, the database table has to be registered.

```
ServiceSchema:registerCalculatedColumn("Customer","AddressLines","character").
```

The calculation method is dynamically invoked whenever a fill for a table row happens and has to have the name "Calculate<DBTableName>". It's not possible to use a standard fill event handler here since the entire request is dynamic and there is no static definition available.

```
METHOD PUBLIC VOID CalculateCustomer( phBuffer AS HANDLE, pcColumns AS CHARACTER):
    DEFINE VARIABLE cLines AS CHARACTER NO-UNDO.
    IF LOOKUP("AddressLines",pcColumns) > 0 THEN
    DO:
        ASSIGN cLines = phBuffer::NAME + CHR(10) + FormattedAddress(phBuffer).
        phBuffer::AddressLines = cLines.
    END.
    RETURN.
END METHOD.
```

If a column isn't used in a request then it's not available in the buffer (temp-table). All other columns from the DB table are there. If you want to use static code in your calculation methods then load the DB table in a buffer and write code for the DB table.

```
FIND Customer WHERE Customer.CustNum = phBuffer::CustNum NO-LOCK.
...
```

Another example for a calculated column would be the order total for the Sports2000 order table. In the OrderLine table there is an ExtendedPrice but no calculated OrderTotal in the Order table.

```
ServiceSchema:registerCalculatedColumn("Order","OrderTotal","decimal").
```

```
METHOD PUBLIC VOID CalculateOrder( phBuffer AS HANDLE, pcColumns AS CHARACTER):
    DEFINE VARIABLE dTotal AS DECIMAL NO-UNDO.
    IF LOOKUP("OrderTotal",pcColumns) > 0 THEN
    DO:
        FIND Order WHERE Order.Ordernum = phBuffer::OrderNum NO-LOCK NO-ERROR.
        FOR EACH OrderLine OF Order NO-LOCK:
            dTotal = dTotal + OrderLine.ExtendedPrice.
        END.
        phBuffer::OrderTotal = dTotal.
    END.
    RETURN.
END METHOD.
```

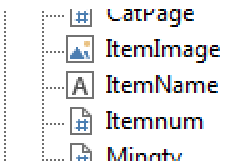
You can do more complex calculations outside the service in other classes or procedures if you want.

Calculated columns can also be used to send images from the service to List & Label. Images are defined as blob's and whenever you use blobs you should define the mime type:

```
ServiceSchema:registerCalculatedColumn("Item","ItemImage","blob","image/jpeg").
```

```
METHOD PUBLIC VOID CalculateItem( phBuffer AS HANDLE, pcColumns AS CHARACTER):
    DEFINE VARIABLE cLines AS CHARACTER NO-UNDO.
    DEFINE VARIABLE cFile  AS CHARACTER NO-UNDO.
    DEFINE VARIABLE hField AS HANDLE    NO-UNDO.
    DEFINE VARIABLE m      AS MEMPTR    NO-UNDO.
    IF LOOKUP("ItemImage",pcColumns) > 0 THEN
    DO:
        cFile = SUBSTITUTE("ListLabelDemo/Images/cat&1.jpg", STRING(phBuffer::ItemNum,"99999")).
        FILE-INFO:FILE-NAME = cFile.
        IF FILE-INFO:FULL-PATHNAME <> ? THEN
        DO:
            COPY-LOB FROM FILE FILE-INFO:FULL-PATHNAME TO m.
            phBuffer::ItemImage = m.
            SET-SIZE(m) = 0.
        END.
    END.
    RETURN.
END METHOD.
```

The images are recognized as drawing in List & Label and you can just add them to a layout.



CatPage
ItemImage
ItemName
Itemnum
Minqty

Itemnum	ItemName	ItemImage
1	Fins	
2	Tennis Racquet	
3	Golf Umbrella	

Native Aggregate Functions

List & Label has support for Native Aggregate Functions. Instead of reading many from the database to do a calculation inside List & Label, these calculations can be performed on the data provider / server.

In the SQL world these aggregations are part of the language and quite easy to implement. An example is: How much revenue has a sales person generated.

```
select sum(OrderTotal) from order where  order.salesrep = "BBB”.
```

In pseudo code what List & Label implement looks like this:

```
select [distinct] <function> (<expression>) from <table> where <filter>.
```

The result is a single value but with different data types.

In OpenEdge the hard part is the expression. Normally this would be a single column but could also be a calculation like "(column-a + column-b) * 2”.

Open Edge doesn't support dynamic calculation. Therefore, a solution based on the shunting yard algorithm (see https://en.wikipedia.org/wiki/Shunting-yard_algorithm) was implemented. This is a two-phase process. First the expression has to be transformed into RPN (reverse polish notation) and then the RPN has to be evaluated with given values or in our case column names. Have a look at ListLabel.OpenEdgeAdapter.OpenEdgeNativeFunction.cls to see how it works.

There are currently some limitations:

- The result of a native function is always decimal.
- DISTINCT is not supported in the moment (ignored).

Testing the service

You need three things:

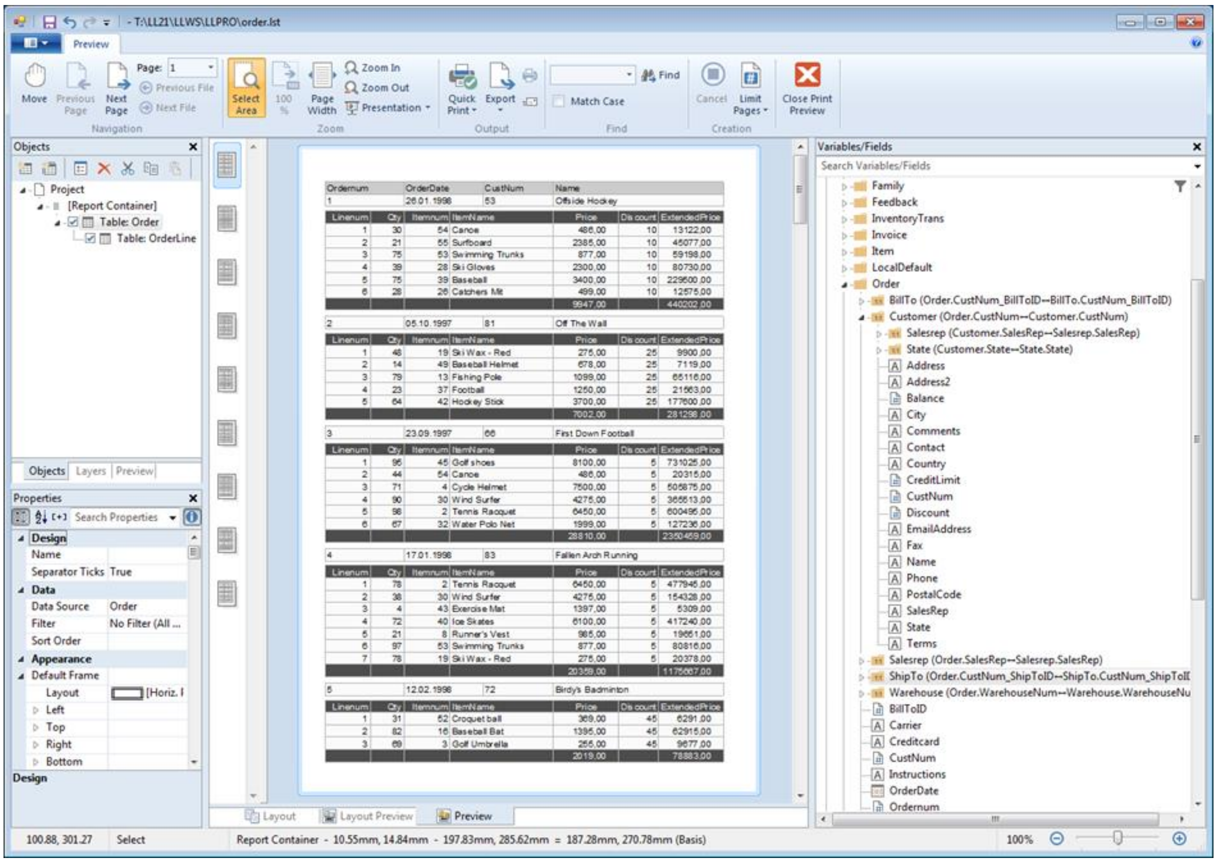
- ListLabel
- OpenEdgeDataProvider
- A service

```
USING TasteITConsulting.ListLabel29.OpenEdgeDataProvider FROM ASSEMBLY.
USING combit.ListLabel29.ListLabel FROM ASSEMBLY.
USING ListLabelDemo.Sports2000ServiceAdapter FROM PROPATH.

DEFINE VARIABLE oProvider       AS OpenEdgeDataProvider     NO-UNDO.
DEFINE VARIABLE oLL             AS ListLabel                NO-UNDO.
DEFINE VARIABLE oServiceAdapter AS Sports2000ServiceAdapter NO-UNDO.

oProvider = NEW OpenEdgeDataProvider().
oServiceAdapter = NEW Sports2000ServiceAdapter().
oLL = NEW ListLabel().

/* Get the schema */
oProvider:ServiceAdapter = oServiceAdapter.
oProvider:ServiceName    = "ListLabelDemo.Sports2000Service".
oProvider:Initialize().

/* Optional: Limit designer queries */
oProvider:MaxRows = 50.

/* Optional: Limit the size of the schema, if you already know the master
table */
oLL:DataMember = "Order".

oLL:DataSource      = oProvider.
oLL:ForceSingleThread = TRUE.
oLL:Design().
oLL:Dispose().
```

This starts the List & Label Designer and you have access to your entire database.

It's recommended to set the MaxRows property if you are running the designer. This limits queries in the designer preview and your service only returns a maximum of MaxRows rows.

If you already know the Master Table for your report then you can also set the DataMember for List & Label. List & Label then limits the schema to this table and all other tables having relations to this table.



Design	
Name	
Separator Ticks	True
Data	
Data Source	Order
Filter	No Filter (All ...
Sort Order	
Appearance	
Default Frame	
Layout	[Horiz. I
Left	
Top	
Right	
Bottom	

Design

Ordernum	OrderDate	CustNum	Name
1	26.01.1998	53	Offside Hockey

Linenum	Qty	Itemnum	ItemName	Price	Discount	ExtendedPrice
1	30	54	Canoe	486,00	10	13122,00
2	21	55	Surfboard	2385,00	10	45077,00
3	75	53	Swimming Trunks	877,00	10	59198,00
4	39	28	Ski Gloves	2300,00	10	80730,00
5	75	39	Baseball	3400,00	10	229500,00
6	28	26	Catchers Mit	499,00	10	12575,00
				9947,00		440202,00

2	05.10.1997	81	Off The Wall

Linenum	Qty	Itemnum	ItemName	Price	Discount	ExtendedPrice
1	48	19	Ski Wax - Red	275,00	25	9900,00
2	14	49	Baseball Helmet	678,00	25	7119,00
3	79	13	Fishing Pole	1099,00	25	65116,00
4	23	37	Football	1250,00	25	21563,00
5	64	42	Hockey Stick	3700,00	25	177600,00
				7002,00		281298,00

3	23.09.1997	66	First Down Football

Linenum	Qty	Itemnum	ItemName	Price	Discount	ExtendedPrice
1	96	45	Golf shoes	8100,00	5	731025,00
2	44	54	Canoe	486,00	5	20316,00
3	71	4	Cycle Helmet	7500,00	5	505875,00
4	90	30	Wind Surfer	4275,00	5	365513,00
5	98	2	Tennis Racquet	6450,00	5	600495,00
6	67	32	Water Polo Net	1999,00	5	127236,00
				28810,00		2350459,00

4	17.01.1998	83	Fallen Arch Running

Linenum	Qty	Itemnum	ItemName	Price	Discount	ExtendedPrice
1	78	2	Tennis Racquet	6450,00	5	477945,00
2	38	30	Wind Surfer	4275,00	5	154328,00
3	4	43	Exercise Mat	1397,00	5	5309,00
4	72	40	Ice Skates	6100,00	5	417240,00
5	21	8	Runner's Vest	985,00	5	19651,00
6	97	53	Swimming Trunks	877,00	5	80816,00
7	78	19	Ski Wax - Red	275,00	5	20378,00
				20359,00		1175667,00

5	12.02.1998	72	Birdy's Badminton

Linenum	Qty	Itemnum	ItemName	Price	Discount	ExtendedPrice
1	31	52	Croquet ball	369,00	45	6291,00
2	82	16	Baseball Bat	1395,00	45	62915,00
3	69	3	Golf Umbrella	255,00	45	9677,00
				2019,00		78883,00

100.88, 301.27 Select Report Container - 10.55mm, 14.84mm - 197.83mm, 285.62mm = 187.28mm, 270.78mm (Basis)

Sports2000 Service in the designer.

Dataset Services

Dataset services behave like database services accept they are working with in-memory data. You can use these services to pass data from business entities or other client objects. All you need to have is a dataset handle. If you just have a temp-table then create a dataset around it (can be dynamic).

There is a general dataset wrapper class ListLabel.OpenEdgeAdapter.OpenEdgeDatasetService. This class inherits from OpenEdgeService and implements IServiceAdapter.

Here is a sample. It looks a bit different but is easy to use.

```
USING combit.ListLabel21.ListLabel FROM ASSEMBLY.
USING TasteITConsulting.ListLabel21.OpenEdgeDataProvider FROM ASSEMBLY.
USING ListLabel.OpenEdgeAdapter.OpenEdgeDatasetService   FROM PROPATH.

DEFINE TEMP-TABLE ttData NO-UNDO
    FIELD DataNumber AS INTEGER
    FIELD DataText   AS CHARACTER
    INDEX pix IS PRIMARY UNIQUE DataNumber.

DEFINE DATASET dsData FOR ttData.

DEFINE VARIABLE oLL             AS ListLabel NO-UNDO.
DEFINE VARIABLE oProvider       AS OpenEdgeDataProvider   NO-UNDO.
DEFINE VARIABLE oDatasetService AS OpenEdgeDatasetService NO-UNDO.
DEFINE VARIABLE iRow            AS INTEGER NO-UNDO.

DO iRow = 1 TO 100:
    CREATE ttData.
    ASSIGN ttData.DataNumber = iRow
           ttData.DataText   = SUBSTITUTE("Hello world &1",iRow).
END.

oProvider                 = NEW OpenEdgeDataProvider().
oDatasetService           = NEW OpenEdgeDatasetService(DATASET dsData:HANDLE ).
/* Optional - remove prefix like "tt" or "e" */
oDatasetService:TablePrefixToRemove = "tt".

/* The service is also the service adapter */
oProvider:ServiceName    = oDatasetService:ServiceName.
oProvider:ServiceAdapter = oDatasetService.
oProvider:Initialize().

oLL = NEW ListLabel().
oLL:DataSource            = oProvider.
oLL:ForceSingleThread     = TRUE.

oLL:Design().
oLL:Dispose().

RETURN.
```

ABL Demo Application

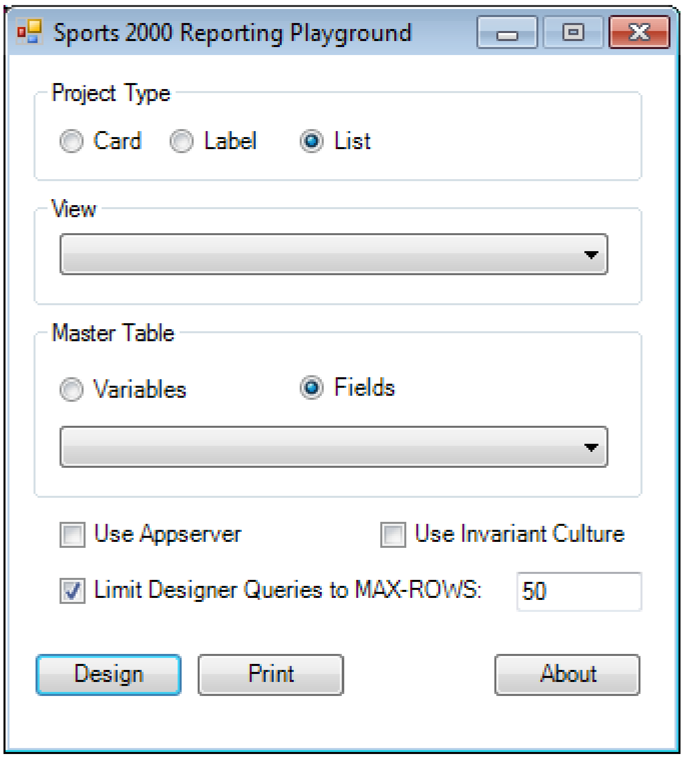
The demo application is a good starting point. You can play around with the sports 2000 service or even use your own service. You may also make a copy of the demo application and change it to match your needs.

- Create your own application from the demo.
- Create your own service adapter.
- Create your own service.

If you decide to use List & Label in combination with a single database service you already have almost everything you need to be able to design all of your documents.

To start the demo either run sports2000demo.p or open the ListLabel.Sports2000DemoForm and launch it.

ListLabelDemo.Sports2000DemoForm



Project Type: Card Label [✓ List]

View:

Master Table: Variables [✓ Fields]

Use Appserver

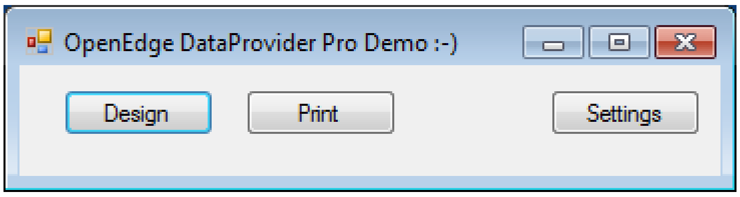
Use Invariant Culture

[✓] Limit Designer Queries to MAX-ROWS: 50

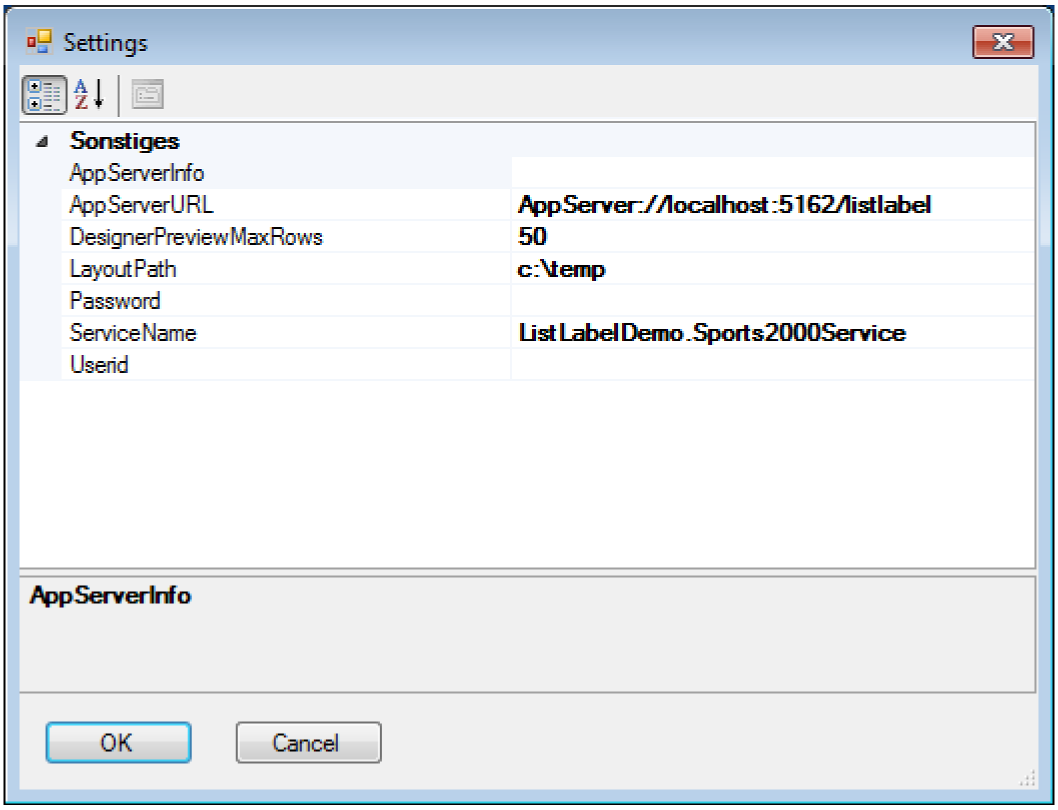
- Type: LL project type
- View: Use a view defined by your service. If empty then the entire schema is used.
- Master Table: You can select a master table from the full schema or the view. List & Label then automatically limits the tables to those being related to the master table.
- Use Appserver: Use the appserver connection defined in the service adapter.
- Use Invariant Culture: If your App Server session doesn't match the client's settings for numeric format and date format then use this flag. The data provider will then pass all strings in American format like "47.11" and "12/31/2016" and the gateway procedure will change the server session during the request.
- Limit designer max rows. Limits queries in the service to return max rows.

.NET Demo Application

The .NET demo is not as complex as the ABL demo, but being able to use .NET as a client is very powerful. You may create a List & Label reporting application that is easily deployed and there is no need to have an Open Edge Client installed.



With the settings screen, you are able to access your own app server and services very easily.



Sonstiges

AppServerInfo	
AppServerURL	AppServer://localhost:5162/listlabel
DesignerPreviewMaxRows	50
LayoutPath	c:\temp
Password	
ServiceName	ListLabelDemo.Sports2000Service
Userid	

AppServerInfo

Passing parameters to a service

A form like the demo is nice for reporting. But there are other documents like single orders. An order form knows about the order id and we have to use this as a filter for the service.

There are two different concepts to pass additional information the service:

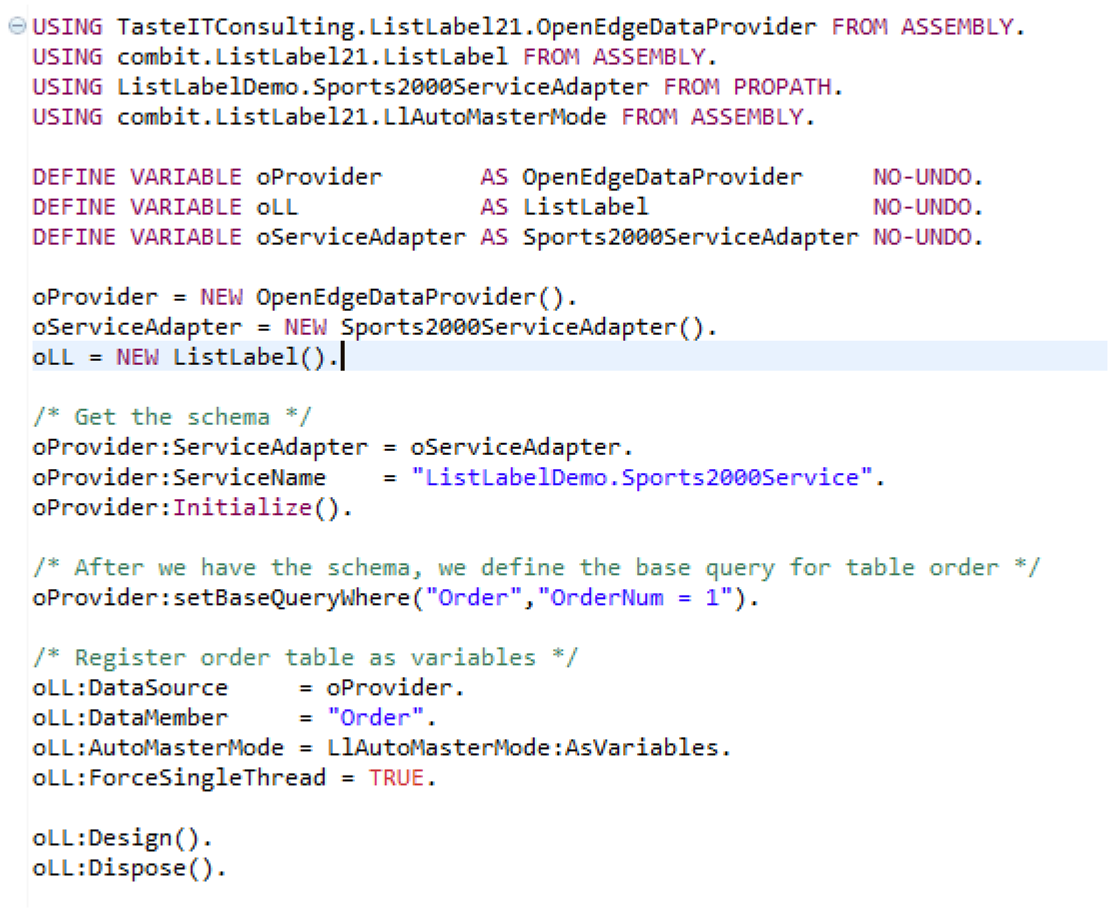
- Service Parameters
- Base Query Filters

Base query filters are defined at table level within the data provider and are added to the query that's build because of the List & Label layout.

Service Parameters are a construct to pass any data to the backend. An example would be something that you need to access your business entities as a service for List & Label.

The standard based on the OpenEdgeService supports Base Query Filters only, but uses Service Parameters to pass additional properties like MaxRows and UseInvariantCulture.

The following is an example to print a single order for a given OrderNum. Table Order registered as variables, OrderLine as fields.



```
USING TasteITConsulting.ListLabel21.OpenEdgeDataProvider FROM ASSEMBLY.
USING combit.ListLabel21.ListLabel FROM ASSEMBLY.
USING ListLabelDemo.Sports2000ServiceAdapter FROM PROPATH.
USING combit.ListLabel21.LlAutoMasterMode FROM ASSEMBLY.

DEFINE VARIABLE oProvider       AS OpenEdgeDataProvider     NO-UNDO.
DEFINE VARIABLE oLL             AS ListLabel                NO-UNDO.
DEFINE VARIABLE oServiceAdapter AS Sports2000ServiceAdapter NO-UNDO.

oProvider = NEW OpenEdgeDataProvider().
oServiceAdapter = NEW Sports2000ServiceAdapter().
oLL = NEW ListLabel().

/* Get the schema */
oProvider:ServiceAdapter = oServiceAdapter.
oProvider:ServiceName    = "ListLabelDemo.Sports2000Service".
oProvider:Initialize().

/* After we have the schema, we define the base query for table order */
oProvider:setBaseQueryWhere("Order","OrderNum = 1").

/* Register order table as variables */
oLL:DataSource     = oProvider.
oLL:DataMember     = "Order".
oLL:AutoMasterMode = LlAutoMasterMode:AsVariables.
oLL:ForceSingleThread = TRUE.

oLL:Design().
oLL:Dispose().
```

Performance

The data provider has been written to be as fast as possible. It makes use of the Used Identifiers meaning that only objects used by a design are registered for printing and that the server also only sends the required fields.

Used Foreign keys of a table are sent together with the table data and cached in the data provider until requested. Each table only requires one app server hit.

When a service sends its own schema to the data provider the schema doesn't only contain definitions to register objects in List & Label. There is also everything included that the data provider needs to formulate an OpenEdgeDataRequest including database table and column names. There is no need that the service builds its own schema for each request just to know what the data sources are.

But you still have to be careful how you design your layout. Let's say you have a nested report with:

- Customer
- Order of customer
- OrderLine of Order, Item of OrderLine (Foreign Key)

Then you have 3 independent tables. List & Label does the following:

- <u>Request</u> Customer.
- Foreach Customer:
    - Print Customer.
        - <u>Request</u> Orders of this customer.
        - For each Order:
            - Print Order.
            - <u>Request</u> OrderLines, Items of this Order.
            - For each OrderLine:
                - Print Orderline, Item.
            - End.
        - End.
- End.

You notice that there are many requests to OpenEdge involved and with mass data this can take forever. There are only to options you have here:

- Create a service that sends all of the data at once like a business entity together with service parameters.
- Change the structure of your layout and use an additional base query.

Structure with only 2 levels:

- Order, Customer of Order as foreign key.
- Orderline of Order, Item of Orderline as foreign key.

So, the key to performance is here to limit the nested table levels.

Another thing to mention is whenever a request is fired from the data provider, the service will send you all the data at once and this is unstoppable! So, playing around with mass data in a production environment can be quite dangerous. Always limit the rows return by setting MaxRows for the designer and even consider setting this property also for printing to a reasonable value of some thousand rows.

Make sure that all of your foreign key (Parent – Child) relations are properly indexed. The child table needs to have an index with leading components of the inherited foreign key.

You cannot prevent the user from choosing any additional column for filtering and sorting. Filtering and sorting is done in the service but may be slow with big and when these columns are not indexed. Make sure that you have a reasonable number of indexes defined for columns being most likely filtered.

Source code

.NET Demo Application

The source code for the .NET Demo Application can be found in the List & Label installation (where this document is located, too). It runs in both OpenEdge versions 32-bit and 64-bit, but needs to be compiled with the right version since the R-Code is no longer portable.

Assemblies

The sample comes with the following 2 assemblies in the "Assemblies" directory:

- combit.ListLabel29.dll (List & Label .NET)
- TasteITConsulting.ListLabel29.dll (C# data provider)

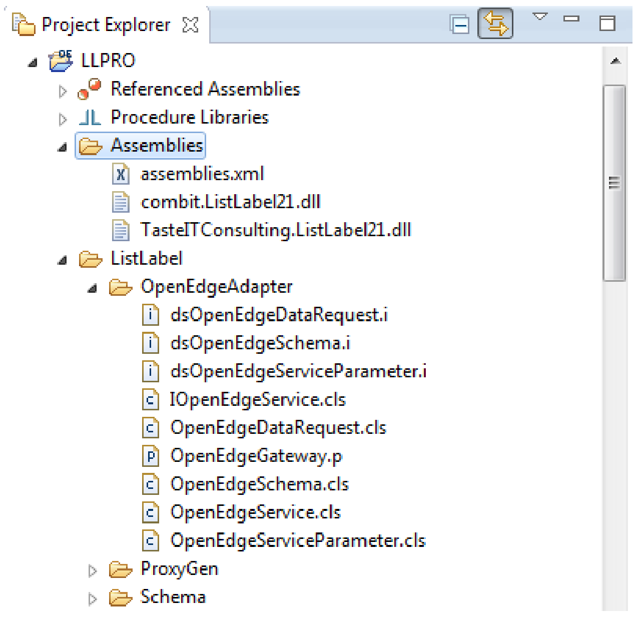
Please note that these 2 assemblies must match. Since the data provider references List & Label and is compiled against a version, you need to rebuild the data provider whenever you use a new version of List & Label (e.g. when installing a List & Label service pack). See chapter "Building the OpenEdgeDataProviderPro Assembly" for that.

Building the Demo Application

After you made sure that the proper assemblies are located in the "Assemblies" directory, you are ready to compile the entire project. In order to build the .NET Demo Application, the right version of the OpenEdge OpenClient DLL needs to be used, too.

- Create a new workspace like C:\LLWS.
- Copy LLPRO to the workspace.
- Start the Developer Studio –data "<your workspace>"
- Use File -> Import -> General -> "Existing Projects into Workspace" and import "LLPRO"
- Make a copy of the sports2000 database and connect it in the project.
- Right click on LLPro and edit properties. Goto "OpenEdge" and change the Startup Parameters to include "-assemblies Assemblies".

The Project Explorer should look like this:



Project Explorer

```
LLPRO
  Referenced Assemblies
  Procedure Libraries
  Assemblies
    assemblies.xml
    combit.ListLabel21.dll
    TasteITConsulting.ListLabel21.dll
  ListLabel
    OpenEdgeAdapter
      dsOpenEdgeDataRequest.i
      dsOpenEdgeSchema.i
      dsOpenEdgeServiceParameter.i
      IOpenEdgeService.cls
      OpenEdgeDataRequest.cls
      OpenEdgeGateway.p
      OpenEdgeSchema.cls
      OpenEdgeService.cls
      OpenEdgeServiceParameter.cls
    ProxyGen
    Schema
```

Then please open ListLabelDemo.Sports2000DemoForm and run it. The demo is a good starting point for your own integration.

If you want to use an app server then please create a new one like "Sports2000" with the following settings:

Broker:

- Stateless
- Application Service Names including: listlabel
  (used in the ListLabelDemo.Sports2000ServiceAdapter)

Agent:

- Server startup parameters: -pf C:\LLWS\LLPRO\sports2000appserver.pf
- Propath: C:\LLWS\LLPRO;@{WinChar Startup\PROPATH};@{WorkPath}

```
# sports2000appserver.pf
# Enter your sports2000 connection here
-db C:\OpenEdge\Databases\sports2000.db
#
-inp 16000
-tok 4000
-rereadnolock
-s 200
-mmax 65534
# Not sure about -Bt
-Bt 2000
```

Data Provider

The source code for the data provider can be obtained on GitHub:
https://github.com/combit/NETDataProviders/tree/main/TasteITConsulting.ListLabel.OpenEdgeDataProvider

Put the "Visual Studio" source code somewhere on the disk and open the solution "OpenEdgeDataProviderPro.sln" in Visual Studio.

The solution includes 2 projects:

- OpenEdgeDataProviderPro – the data provider DLL
- OpenEdgeDemo – A C# demo client with a Progress open client proxy

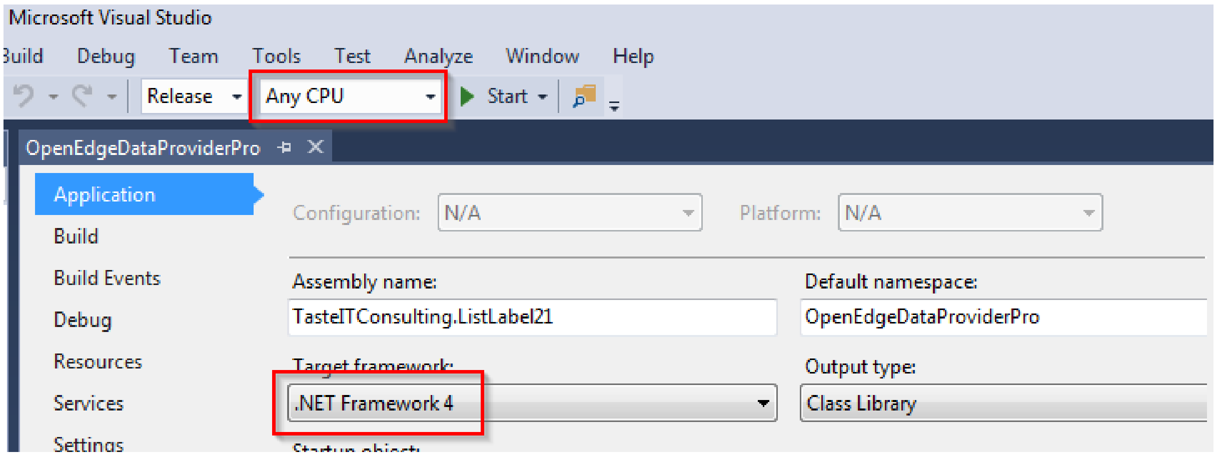
Building the OpenEdgeDataProviderPro Assembly

There a two source files:

- DataProvider.cs (the source code for the dll)
- LitJson.cs (Litjson is a free JSON library, see https://lbv.github.io/litjson/)

References: you need to point to the proper versions of combit.ListLabel29.

Properties: .NET version and CPU

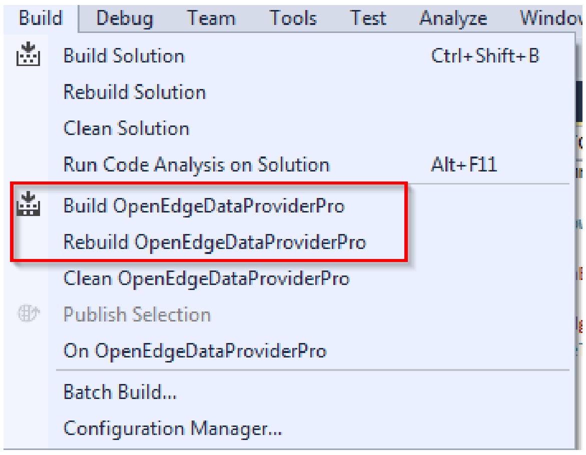


Release | Any CPU

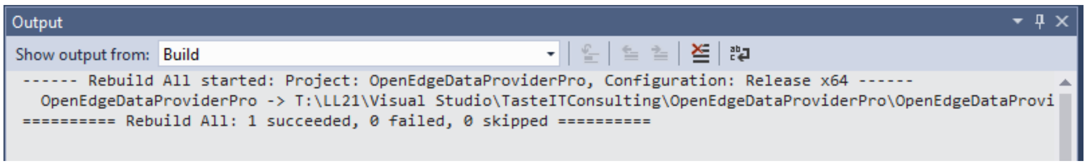
Configuration: N/A
Platform: N/A
Assembly name: TasteITConsulting.ListLabel21
Default namespace: OpenEdgeDataProviderPro
Target framework: .NET Framework 4
Output type: Class Library

Make sure to use "Any CPU”. Then you can use the DLL for 32-bit and 64-bit.

Then you choose a build entry.

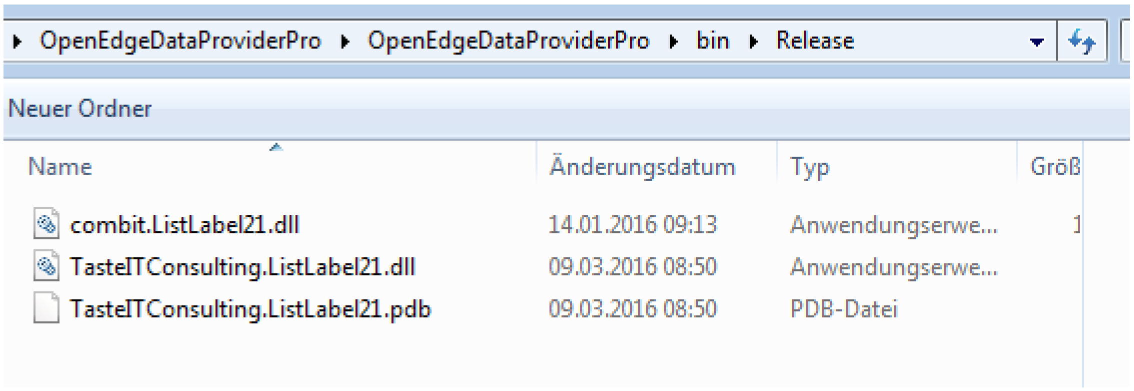

Build
Debug
Team
Tools
Test
Analyze
Windo
Build Solution Ctrl+Shift+B
Rebuild Solution
Clean Solution
Run Code Analysis on Solution Alt+F11
Build OpenEdgeDataProviderPro
Rebuild OpenEdgeDataProviderPro
Clean OpenEdgeDataProviderPro
Publish Selection
On OpenEdgeDataProviderPro
Batch Build...
Configuration Manager...

If you have a proper setup then you should see the following in the output panel:



```
------ Rebuild All started: Project: OpenEdgeDataProviderPro, Configuration: Release x64 ------
  OpenEdgeDataProviderPro -> T:\LL21\Visual Studio\TasteITConsulting\OpenEdgeDataProviderPro\OpenEdgeDataProvi
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

You will find the created DLL in the release directory.


OpenEdgeDataProviderPro ▸ OpenEdgeDataProviderPro ▸ bin ▸ Release
Neuer Ordner

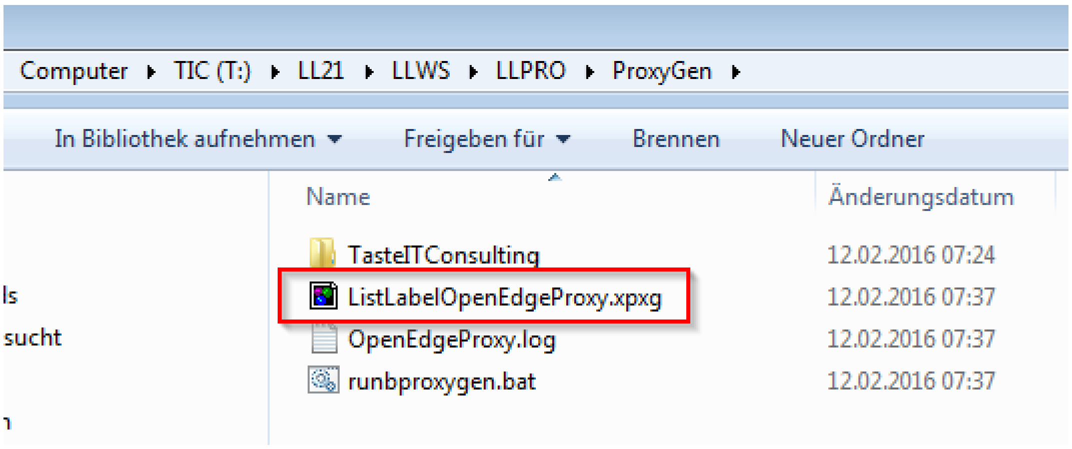
Name	Änderungsdatum	Typ	Größ
combit.ListLabel21.dll	14.01.2016 09:13	Anwendungserwe...	1
TasteITConsulting.ListLabel21.dll	09.03.2016 08:50	Anwendungserwe...	
TasteITConsulting.ListLabel21.pdb	09.03.2016 08:50	PDB-Datei	

Copy these DLLs into your OpenEdge Assemblies directory.

Building the OpenEdgeDemo Application

The .NET demo is a windows forms application accessing the OpenEdge Appserver through an Open Client proxy. The proxy needs to be built with the OpenEdge ProxyGen tool.

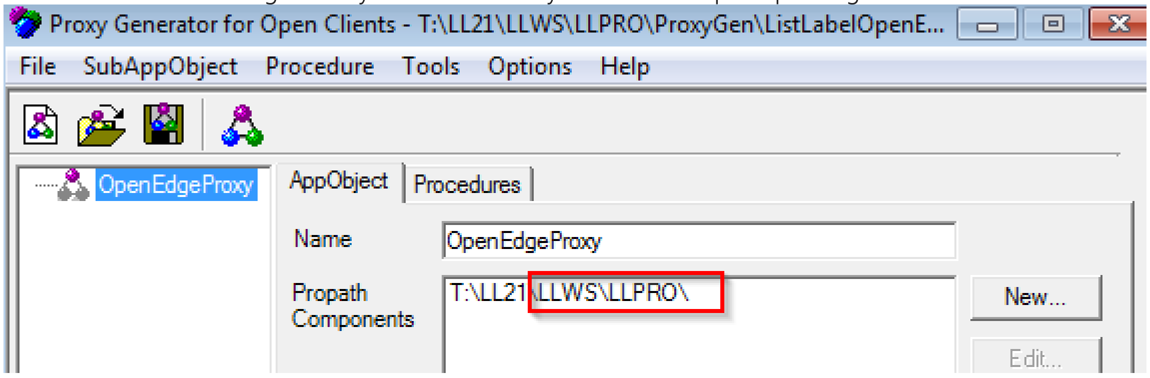
You will find the Proxy Gen configuration in the "ProxyGen" folder of the OpenEdge sample in your installation.



Name	Änderungsdatum
TasteITConsulting	12.02.2016 07:24
ListLabelOpenEdgeProxy.xpxg	12.02.2016 07:37
OpenEdgeProxy.log	12.02.2016 07:37
runbproxygen.bat	12.02.2016 07:37

You may have to change the propath setting to point to the LLPRO directory in your workspace.

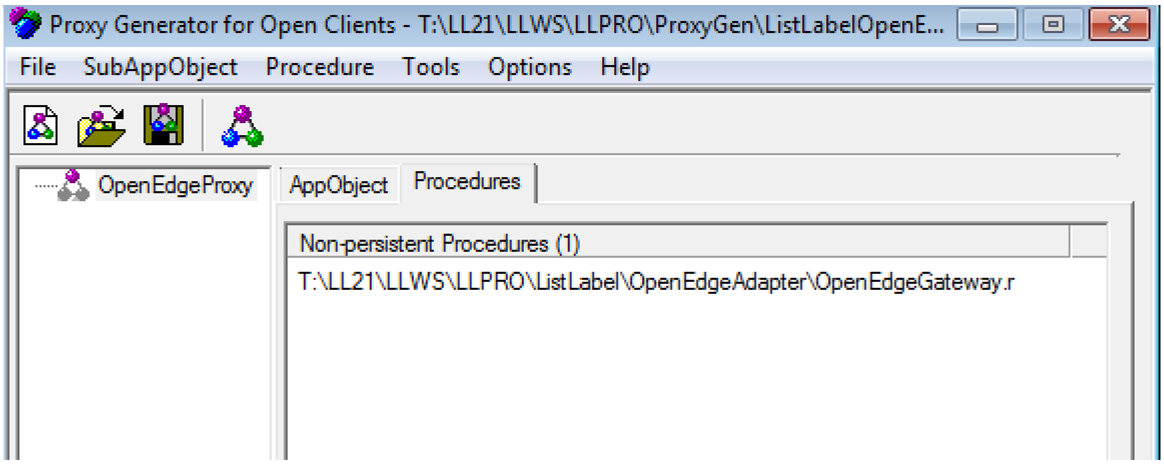
Make sure to use the right ProxyGen shortcut if you have multiple OpenEdge versions installed.



Name: OpenEdgeProxy

Propath Components: T:\LL21\LLWS\LLPRO\

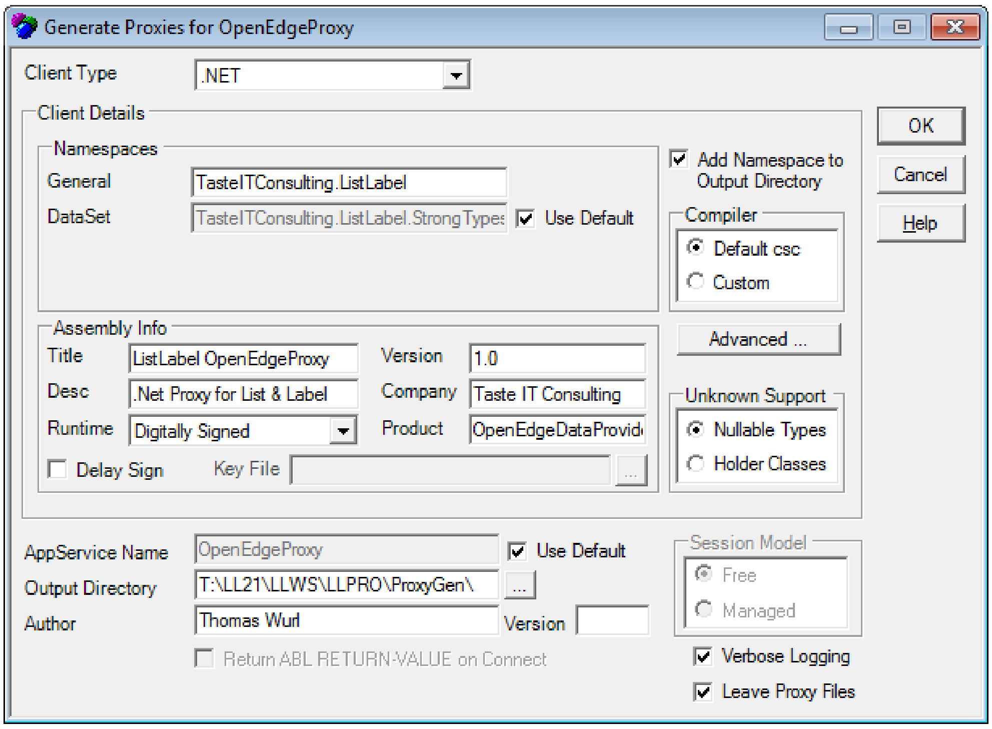
The proxy just needs a single non-persistent procedure OpenEdgeGateway.r handling all the communication.


Proxy Generator for Open Clients - T:\LL21\LLWS\LLPRO\ProxyGen\ListLabelOpenE...
File
SubAppObject
Procedure
Tools
Options
Help
OpenEdgeProxy
AppObject
Procedures

Non-persistent Procedures (1)

T:\LL21\LLWS\LLPRO\ListLabel\OpenEdgeAdapter\OpenEdgeGateway.r

The generator screen should look like this – except the directory.



Generate Proxies for OpenEdgeProxy

Client Type: .NET

Client Details

Namespaces
- General: TasteITConsulting.ListLabel
- DataSet: TasteITConsulting.ListLabel.StrongTypes — [✓] Use Default

[✓] Add Namespace to Output Directory

Compiler: [✓ Default csc] Custom

Assembly Info
- Title: ListLabel OpenEdgeProxy
- Desc: .Net Proxy for List & Label
- Runtime: Digitally Signed
- Version: 1.0
- Company: Taste IT Consulting
- Product: OpenEdgeDataProvid
- [ ] Delay Sign — Key File:

Unknown Support: [✓ Nullable Types] Holder Classes

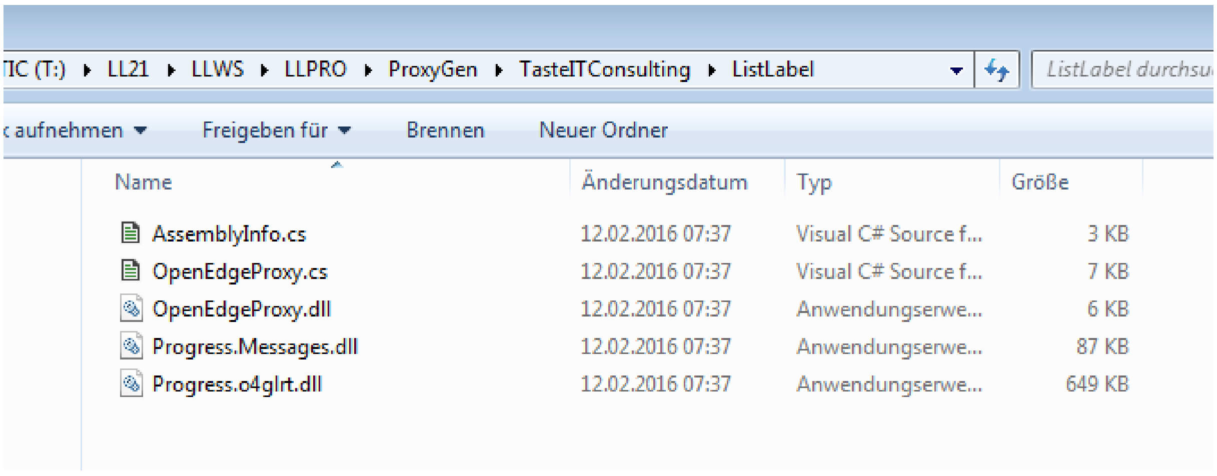
- AppService Name: OpenEdgeProxy — [✓] Use Default
- Output Directory: T:\LL21\LLWS\LLPRO\ProxyGen\
- Author: Thomas Wurl — Version:
- [ ] Return ABL RETURN-VALUE on Connect

Session Model: [✓ Free] Managed

[✓] Verbose Logging

[✓] Leave Proxy Files

After generating the proxy, you will find everything inside the configured output directory:



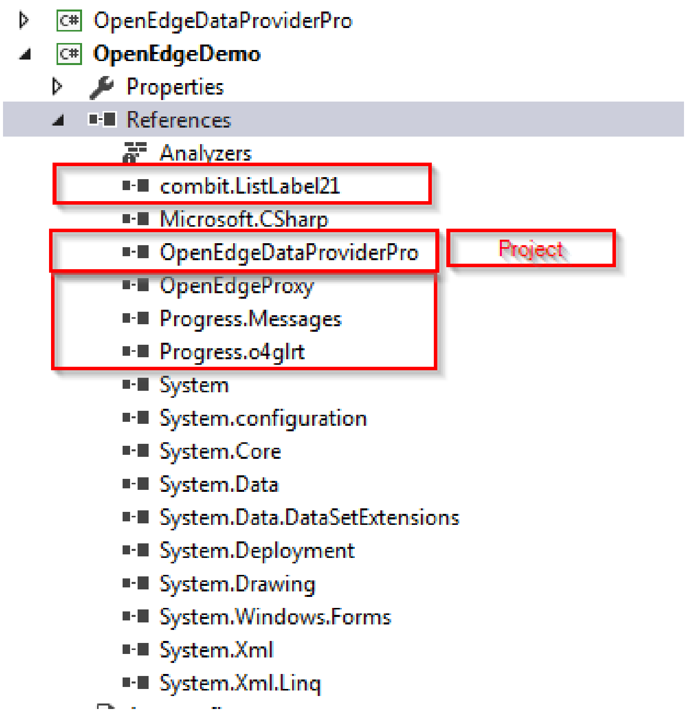
TIC (T:) ▸ LL21 ▸ LLWS ▸ LLPRO ▸ ProxyGen ▸ TasteITConsulting ▸ ListLabel

Name	Änderungsdatum	Typ	Größe
AssemblyInfo.cs	12.02.2016 07:37	Visual C# Source f...	3 KB
OpenEdgeProxy.cs	12.02.2016 07:37	Visual C# Source f...	7 KB
OpenEdgeProxy.dll	12.02.2016 07:37	Anwendungserwe...	6 KB
Progress.Messages.dll	12.02.2016 07:37	Anwendungserwe...	87 KB
Progress.o4glrt.dll	12.02.2016 07:37	Anwendungserwe...	649 KB

Progress adds Progress.Messages.dll and Progress.o4glrt.dll. These DLL come in a 32bit and a 64bit version. If you want to support both, then you have to install both OpenEdge versions.

In our Visual Studio OpenEdgeDemo project we need all of the above DLL and again have to make sure to have the right references and settings for our project to be able to compile it.

For the demo you need to build a 32bit and a 64bit version. Use "X64" instead of "Any CPU" for a 64bit version.



The demo has two forms OpenEdgeDemoForm and SettingsForm.

Inside the OpenEdgeDemoForm is an implementation of the required IOpenEdgeServiceAdapter interface that talks to the OpenEdge Appserver through the generated client proxy.

Closing Remark

Please note, that the OpenEdge Data Provider for List & Label is provided on an "as-is" basis. combit does not offer any support for it. If you need adaptions, you might be able to contact Thomas Wurl, Taste IT Consulting, https://www.taste-consulting.de/, for consulting on how to integrate the data provider into your application or even to customize it to your needs.

combit GmbH   Buecklestr. 3-5   78467 Konstanz   Germany

combit.com