How To Integrate SAP Business Data Into SharePoint 2010 Using Business Connectivity Services And LINQ to SAP

Jürgen Bäurle

August 2010

Parago Media GmbH & Co. KG

Introduction

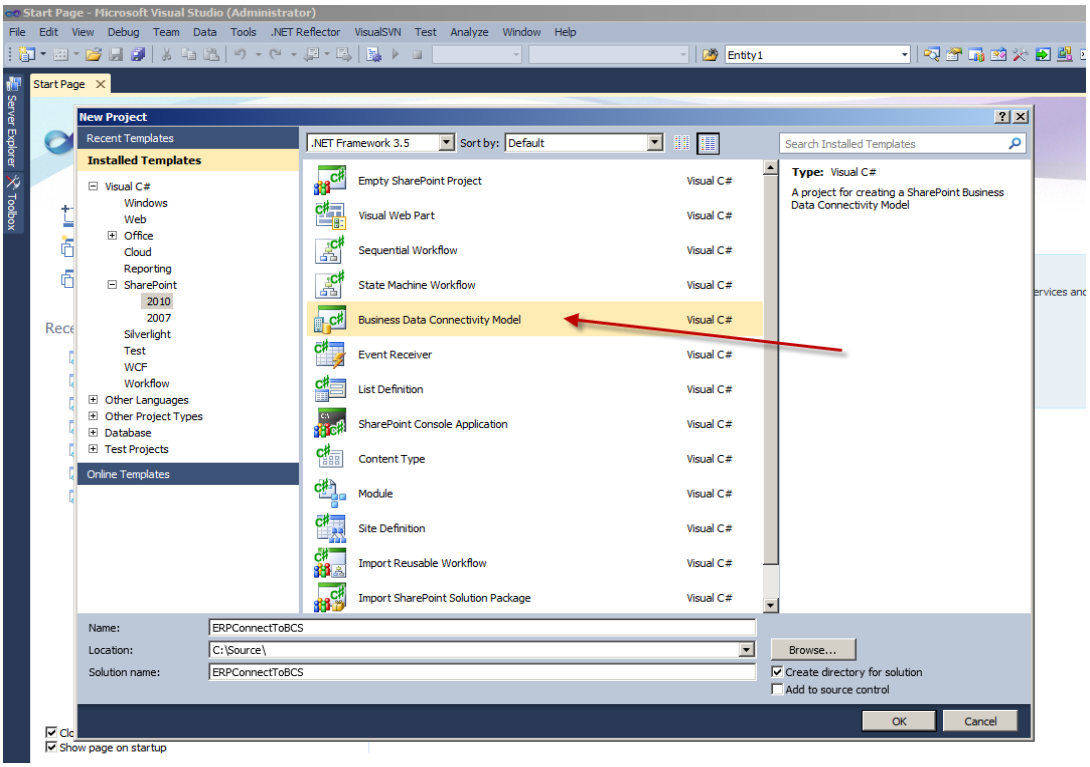
One of the core concepts of the Business Connectivity Services (BCS) for SharePoint 2010 are the external content types. They are reusable metadata descriptions of connectivity information and behaviours (stereotyped operations) applied to external data. SharePoint offers developers several ways to create external content types and integrate them into the platform. The SharePoint Designer 2010 for instance allows you to create and manage external content types that are stored in supported external systems. Such an external system could be SQL Server, WCF Data Service or a .NET Assembly Connector.

This article shows you how to create an external content type for SharePoint named *Customer* based on given SAP customer data. The definition of the content type will be provided as a .NET assembly and the data are displayed in an external list in SharePoint.

The SAP customer data are retrieved from the function module *SD_RFC_CUSTOMER_GET*. In general, function modules in a SAP R/3 system are comparable with public and static C# class methods and can be accessed from outside of SAP via RFC (Remote Function Call). Fortunately we do not need to program RFC calls manually. We will use the very handy *ERPConnect* library from Theobald Software. The library includes a LINQ to SAP provider and designer that makes the our live easier.

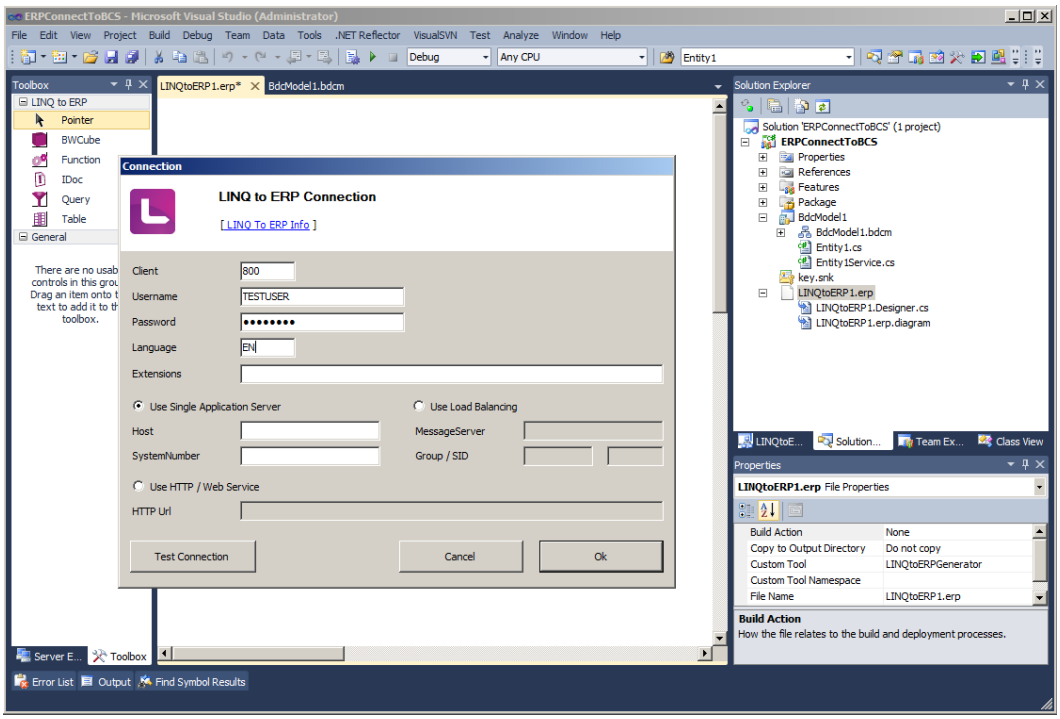
.NET Assembly Connector for SAP

The first step in providing a custom connector for SAP is to create a SharePoint project with the SharePoint 2010 Developer Tools for Visual Studio 2010. Those tools are part of Visual Studio 2010. We will use the *Business Data Connectivity Model* project template to create our project:

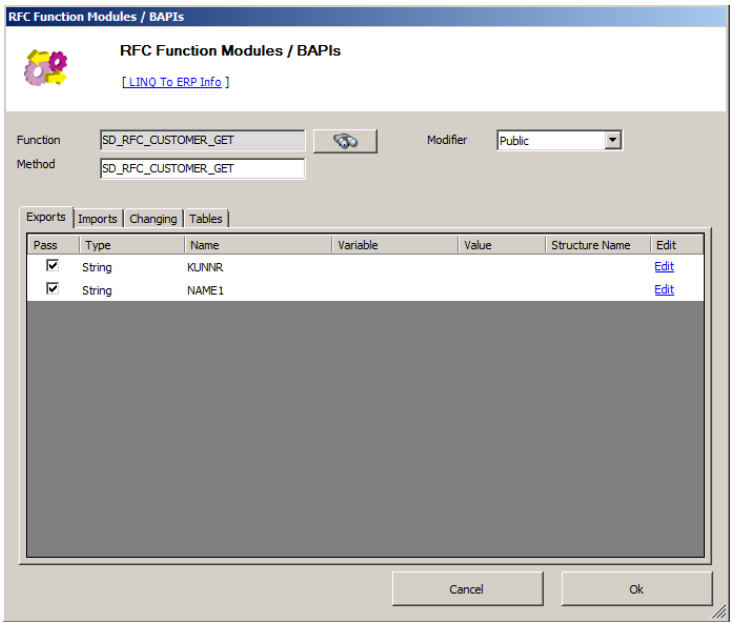


After defining the Visual Studio solution name and clicking the *OK* button, the project wizard will ask what kind of SharePoint 2010 solution you want to create. The solution must be deployed as a farm solution, not as a sandboxed solution. Visual Studio is now creating a new SharePoint project with a default BDC model (*BdcModel1*). You can also create an empty SharePoint project and add a *Business Data Connectivity Model* project item manually afterwards. This will also generate a new node to the Visual Studio Solution Explorer called *BdcModel1*. The node contains a couple of project files: The BDC model file (file extension *bdcm*), the *Entity1.cs* and *EntityService.cs* class files.

Next, we add a LINQ to SAP file to handle the SAP data access logic by selecting the LINQ to ERP item from the *Add New Item* dialog in Visual Studio. This will add a file called *LINQtoERP1.erp* to our project. The LINQ to SAP provider is internally called LINQ to ERP. Double click the LINQtoERP1.erp to open the designer. Now, drag the *Function* object from the designer toolbox onto the design surface. This will open the SAP connection dialog since no connection data have been defined so far:



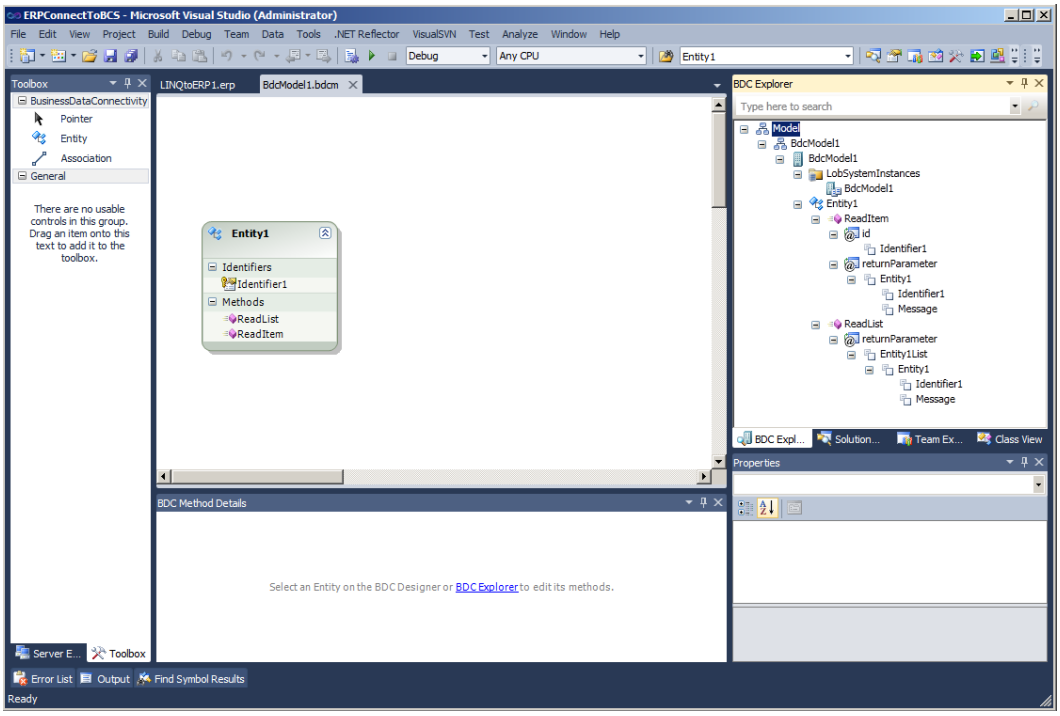
Enter the SAP connection data and your credentials. Click the *Test Connection* button to test the connectivity. If you could successfully connect to your SAP system, click the *Ok* button to open the function module search dialog. Now search for *SD_RFC_CUSTOMER_GET*, then select the found item and click *Ok* to open the *RFC Function Module / BAPI* dialog:



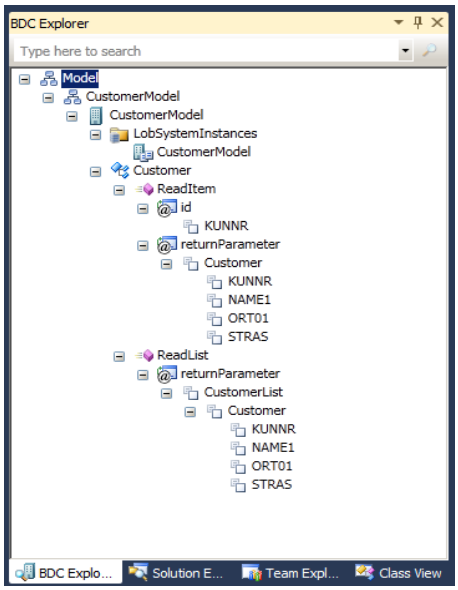
The dialog provides you the option to define the method name and parameters you want to use in your SAP context class. The context class is automatically generated by the LINQ to SAP designer including all SAP objects defined. Those objects are either C# (or VB.NET) class methods and/or additional object classes used by the methods.

For our project we need to select the export parameters *KUNNR* and *NAME1* by clicking the checkboxes in the *Pass* column. These two parameters become our input parameters in the generated context class method named *SD_RFC_CUSTOMER_GET*. We also need to return the customer list for the given input selection. Therefore we select the table parameter *CUSTOMER_T* on the *Tables* tab. Then, click the *Ok* button on the dialog and the new objects gets added to the designer surface.

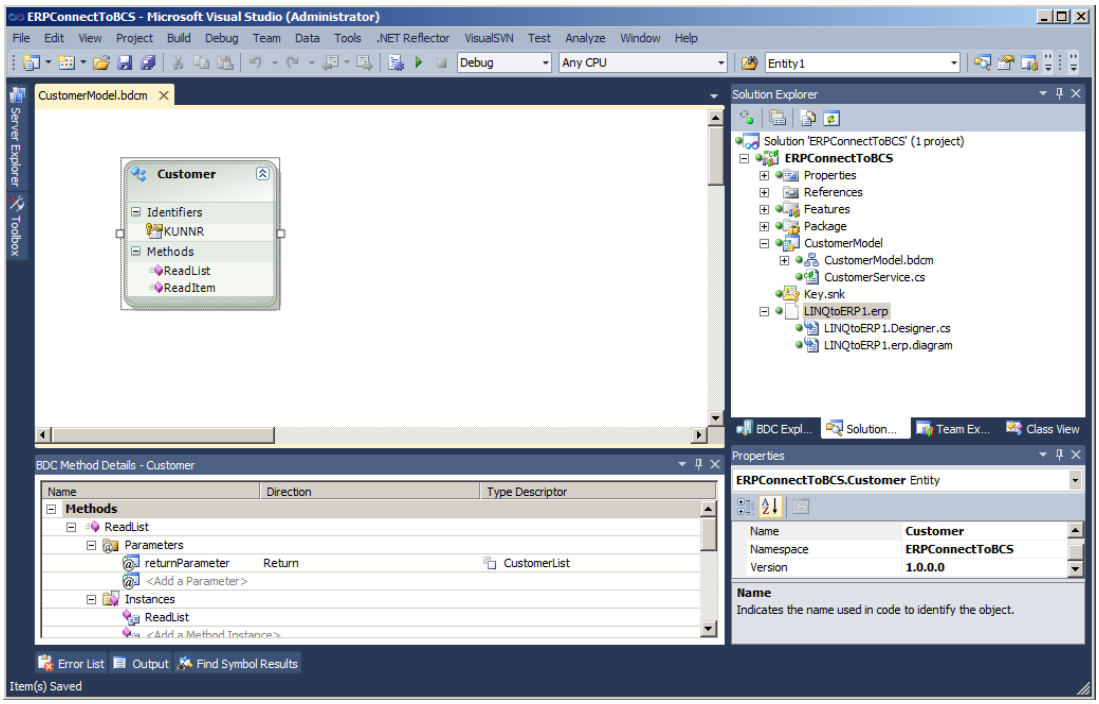
The LINQ designer has also automatically generated a class called *Customer* within the *LINQtoERP1.Designer.cs* file. This class will become our BDC model entity or external content type. But first we need to adjust and rename our BDC model that was created by default from Visual Studio. Currently the BDC model looks like this:



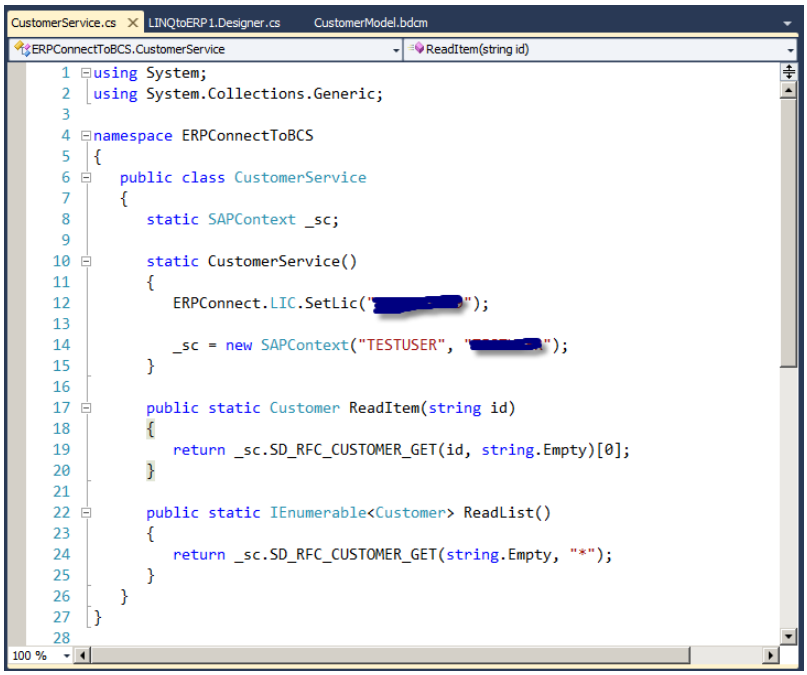
Rename the *BdcModel1* node and file into *CustomerModel*. Since we already have an entity class (*Customer*), delete the file *Entity1.cs* and rename the *EntityService.cs* file into *CustomerService.cs*. Next, open the *CustomerModel* file and rename the designer object Entity1. Then, change the entity identifier name from *Identifier1* to *KUNNR*. You can also use the BDC Explorer for renaming. The final adjustment result should look as follows:



After those modifications the current SharePoint project should look similar to the next screenshot:



The last step we need to do in our Visual Studio project is to change the code in the *CustomerService.cs* class. The BDC model methods *ReadItem* and *ReadList* must be implemented using the automatically generated LINQ to SAP code. First of all, take a look at the code:



As you can see we basically have just a few lines of code. All of the SAP data access logic is encapsulated within the SAP context class (see *LINQtoERP1.Designer.cs* file). The *CustomerService* class just implements a static constructor to set the *ERPConnect* license key and to initialize the static variable *_sc* with the SAP credentials as well as the two BDC model methods.

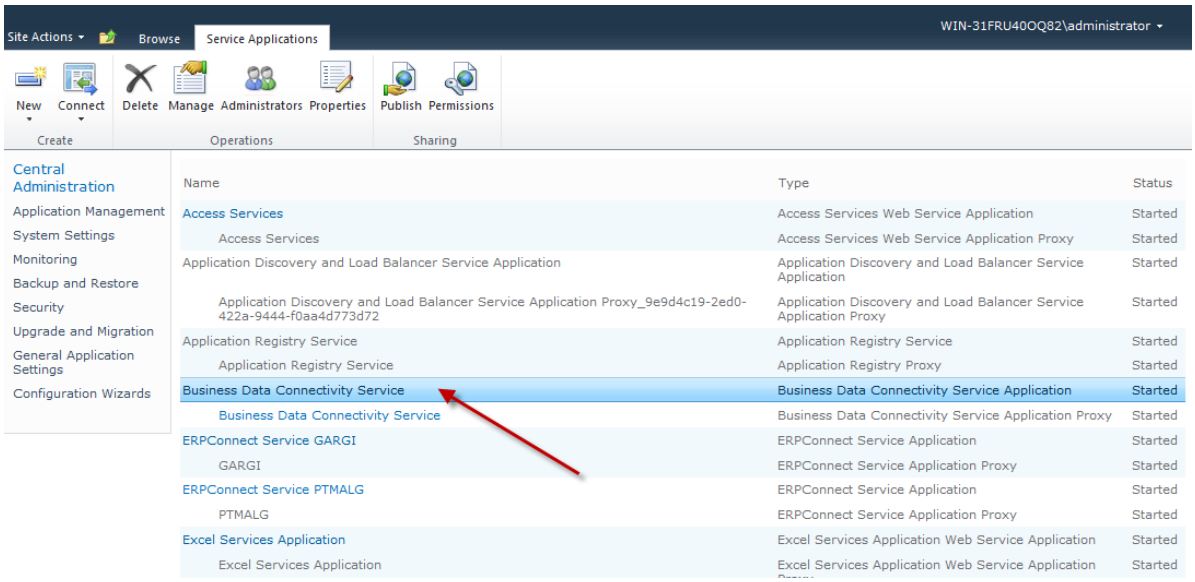
The *ReadItem* method, BCS stereotyped operation *SpecificFinder*, is called by BCS to fetch one specific item defined by the identifier *KUNNR*. In this case we just call the *SD_RFC_CUSTOMER_GET* context method with the passed identifier (variable *id*) and return the first customer object get from SAP.

The *ReadList* method, BCS stereotyped operation *Finder*, is called by BCS to return all entities. In this case we just return all customer objects the *SD_RFC_CUSTOMER_GET* context method returns. The returned result is already of type *IEnumerable<Customer>*.

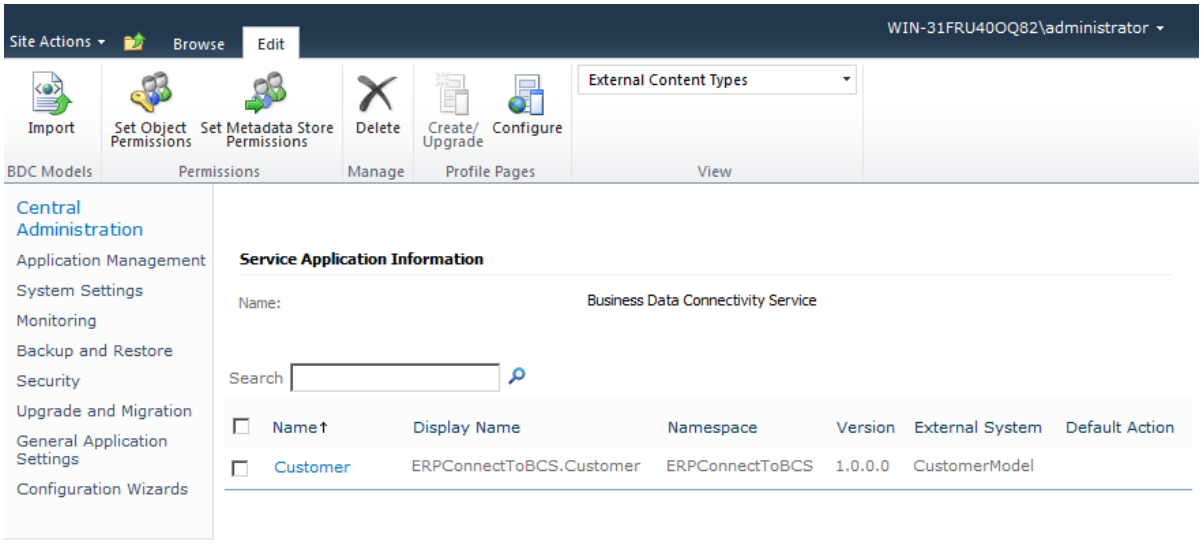
The final step is to deploy the SharePoint solution. Right-click on the project node in the Visual Studio Solution Explorer and select *Deploy*. This will install and deploy the SharePoint solution on the server. You can also debug your code by just setting a breakpoint in the *CustomerService* class and executing the project with F5.

That's all we have to do!

Now, start the SharePoint *Central Administration* panel and follow the link "Manage Service Applications" or navigate directly to the URL *http://<SERVERNAME>/_admin/ServiceApplications.aspx*.

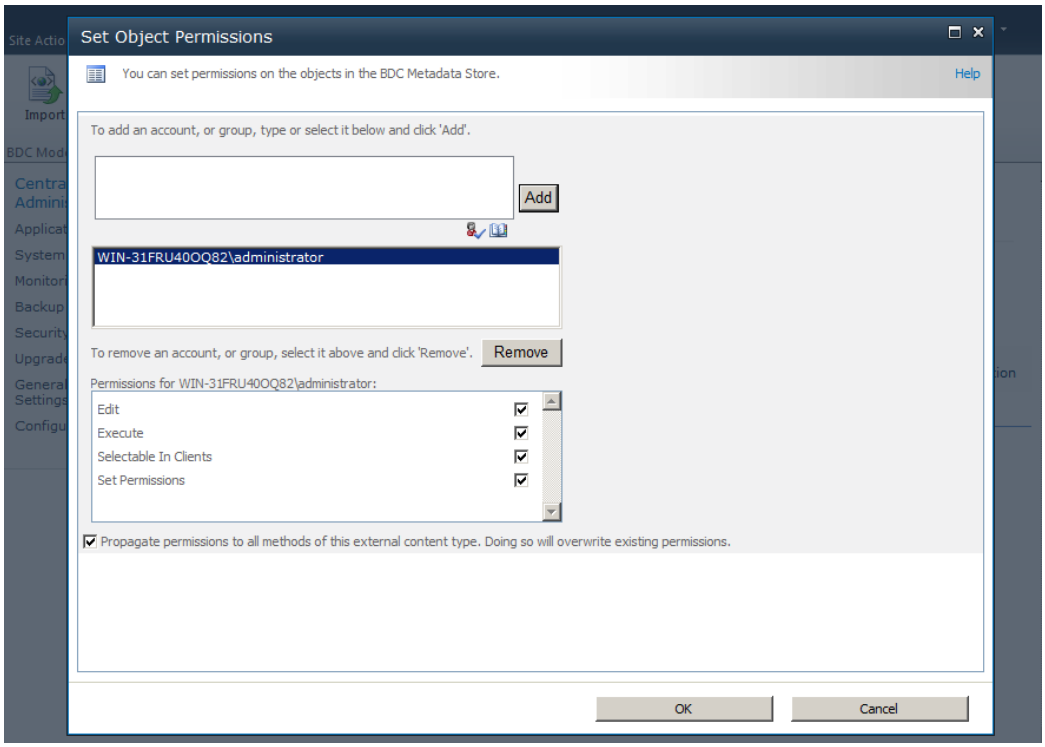


Click on *Business Data Connectivity Service* to show all available external content types:



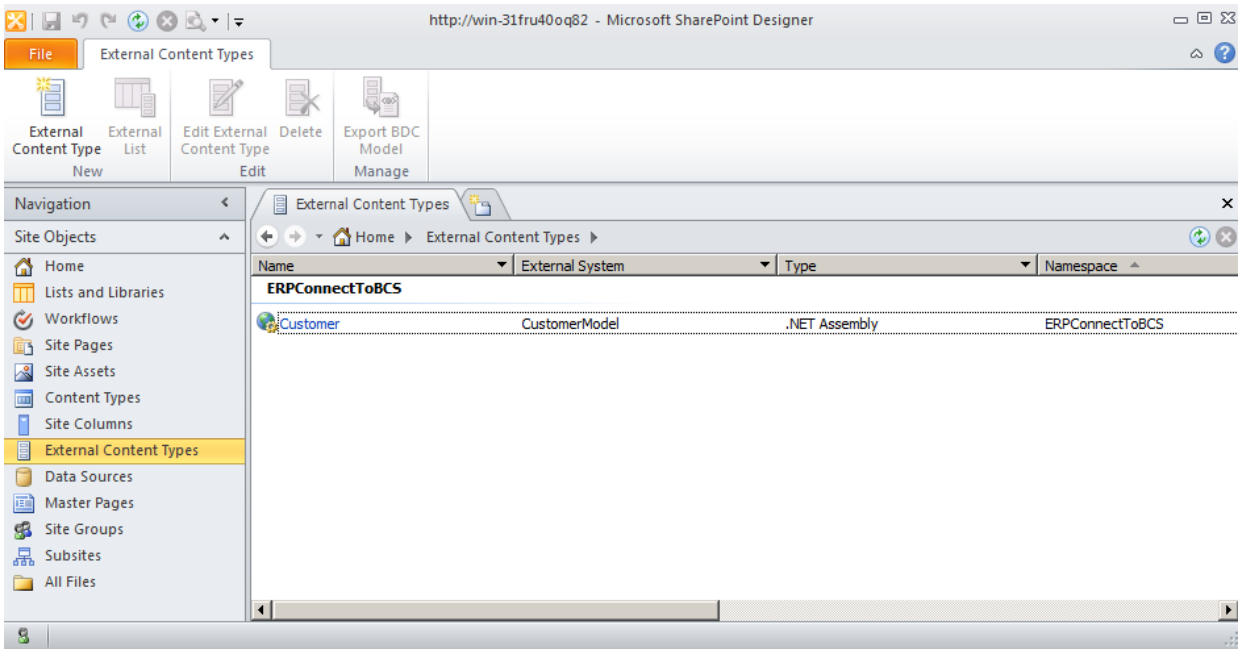
On this page we find our deployed BDC model including the *Customer* entity. You can click on the name to retrieve more details about the entity. Right now, there is just one issue open. We need to set permissions!

Mark the checkbox for our entity and click on the *Set Object Permissions* in the Ribbon menu bar. This will open the following permission dialog:



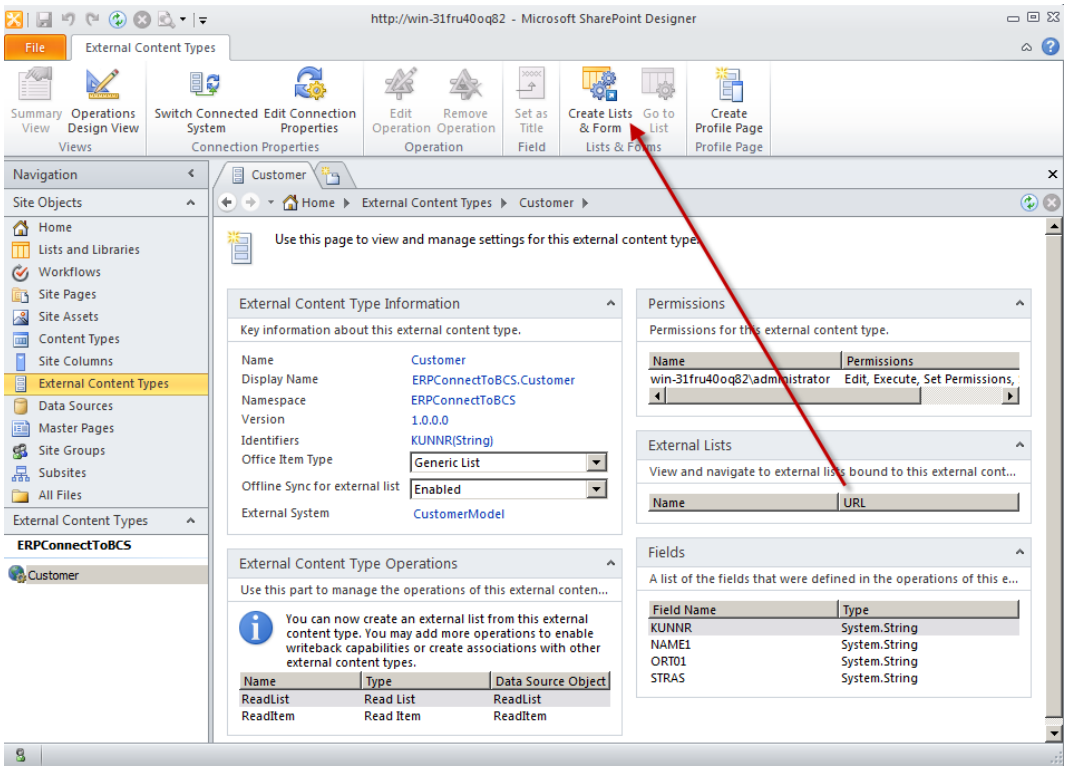
Now, define the permissions for the users you want to allow to access the entity and click the *Ok* button. In the screen shown above the user administrator has all permissions possible.

In the next and final step we will create an external list based on our entity. To do this we open SharePoint Designer 2010 and connect us with the SharePoint web site.



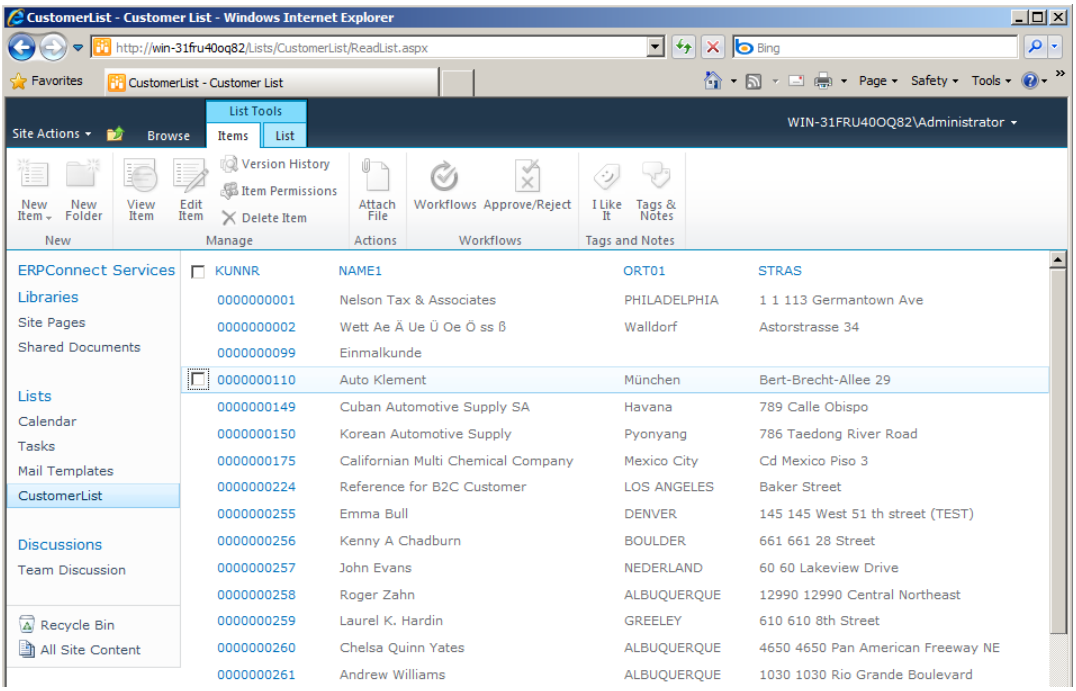
Click on *External Content Types* in the *Site Objects* panel to display all content types (see above). Double click on the *Customer* entity to open the details. The SharePoint Designer is reading all information available by BCS.

In order to create a external list for our entity click on *Create Lists & Form* on the Ribbon menu bar (see screenshot below) and enter *CustomerList* as name for the external list.



Ok, now we are done!

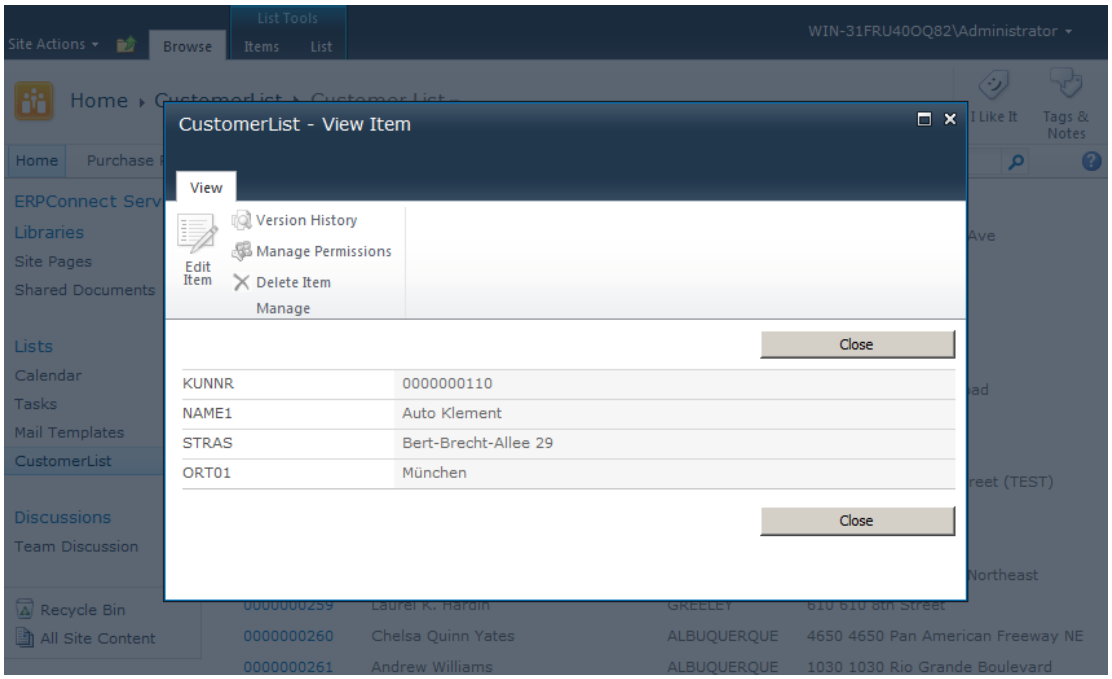
Open the list and you should get the following result:



The external list shows all defined fields for our entity, even though our *Customer* class, automatically generated by the LINQ to SAP, has more than those four fields. This means you can only display a subset of information for your entity.

Another option is to just select those fields required within the LINQ to SAP designer. With the LINQ designer you can access not only SAP function modules. You can integrate other SAP objects, like tables, BW cubes, SAP Query or IDOCs. A demo version of the *ERPConnect* library can be downloaded from the Theobald Software homepage.

If you click the associated link of one of the customer numbers in the column KUNNR (see screenshot above), SharePoint will open the details view:



Summary

This article has shown how easy and simple it is to integrate business data from SAP into the SharePoint platform using standard tools. Combing the powerful Microsoft Visual Studio 2010 with its SharePoint development tools and the handy LINQ to SAP provider tool from Theobald Software, you just need to write a couple of code lines to stick together all the logic we need to create an external list in SharePoint 2010 and BCS.

Contact Information

If you have any feedback or suggestions, please feel free to contact me:

Jürgen Bäurle
jbaurle@parago.de
http://www.parago.de/jbaurle

Parago Media GmbH & Co. KG
Im Wengert 3 | 71336 Waiblingen, Germany | Phone +49.7146.861803 | Internet http://www.parago.de