Gimmal

Information Management for Everyone®

Administration Guide

ERP-Link Modeler

Version 5.4.0

March 2019

Title: *ERP-Link Modeler V5.4.0 Administration Guide*

If you have questions or comments about this publication, please email TechnicalPublications@Gimmal.com. Be sure to identify the guide, version number, section, and page number to which you are referring. Your comments are welcomed and appreciated.

Contents

Introduction

Gimmal delivers market-leading content governance and compliant records solutions built on Microsoft® SharePoint®. Gimmal solutions drive user adoption and simplify information access by making information lifecycle content management simple and transparent. This ensures consistent, enterprise-wide compliance and proactive litigation readiness while lowering costs.

Gimmal's ERP-Link platform includes a tool named the ERP-Link Modeler, which is used for building .Net libraries that execute SAP functions using the Gimmal Connection Service. The libraries that are generated allow for integration of custom solutions with SAP.

Intended Audience

This document is intended for .NET programmers and architects who are designing and implementing a solution to connect SAP to Microsoft products. Familiarity with C# or .NET programming and the Gimmal Connection Service is required to fully understand this document.

Intended Usage

The ERP-Link Modeler is intended to be connected to a development SAP environment for generating code. The tool is not intended to be connected to production environments for performing code generation. As with any custom code, the generated code should be properly tested against a non-production environment prior to deployment for usage in a production capacity.

Overview

The ERP-Link Modeler is a standalone application that allows a developer to browse SAP objects and function modules and create usable C# or Visual Basic.Net classes that can be used in the developer's project to access the functionality of an existing SAP system. This code is in the form of a text snippet that the developer can copy and paste into a project.

ERP-Link Modeler offers a simplified approach that reuses existing SAP framework, services, application objects, and the Microsoft .NET platform.

- No SAP-side changes required to use ERP-Link Modeler or application code generated by it
- No duplicate data store required
- No duplicate business logic required
- Reuses and reflects SAP security
- Leverages Microsoft platform technologies, including Microsoft.NET and Web Services, and Visual Studio IntelliSense

About ERP-Link Objects

ERP-Link Modeler is a standalone program that allows a developer to easily generate **ERP-Link Objects**. ERP-Link Objects are Microsoft .NET-compatible classes that can be used in your C# or Visual Basic.NET projects to access the functionality of an existing SAP System.

The ERP-Link Modeler supports the creation of four different kinds of ERP-Link objects:

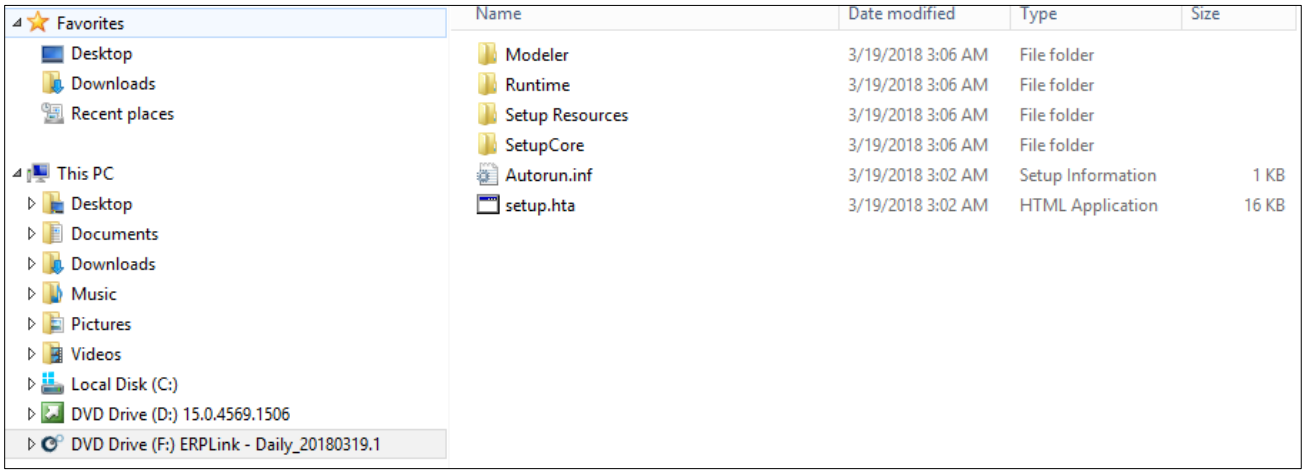
- **ERP-Link business objects** based on the Business Application Programming Interfaces (BAPIs) and Remote Function Call (RFC) modules of an existing SAP System. The generated ERP-Link business objects can then be used in C# or Visual Basic.NET projects to conveniently access the functionality available from the SAP System, without involving the programmer with tedious data conversion details.
- **ERP-Link table view objects** support the reading of SAP relational tables and presenting the data as an ADO.NET System.Data.DataTable, or reading the data into a System.Data.DataSet.
- **ERP-Link query objects** support the invocation of an SAP query and the reading of the returned data through an implementation of the ADO.NET System.Data.IDataReader interface.
- **ERP-Link report objects** support the invocation of SAP Report programs and the reading of the returned data through an implementation of the ADO.NET System.Data.IDataReader interface.

An ERP-Link object, when present in a C# or Visual Basic.NET project, will automatically generate source code into a C# or Visual Basic.NET class, referred to as the **proxy**. This proxy source code is then compiled along with any other source code files. The resulting proxy code can be called from other parts of your projects as a straightforward .NET class.

Installation

ERP-Link Modeler is distributed as a Microsoft Installer package.

1. Download the installer package from the Gimmal download site.
2. Right-click the package in Windows Explorer and select **Run as Administrator**.
3. Select the directory and hard drive that you want to install the product to. The default setting is to install the software in your **Program Files** folder, normally found on the **C:** drive.
4. Click **OK**.
5. Navigate to the installation folder and double-click the "setup.hta" file.



The Gimmal splash screen opens.

6. Under the Install ERP-Link section, click the **Modeler** link to launch the Modeler installer.

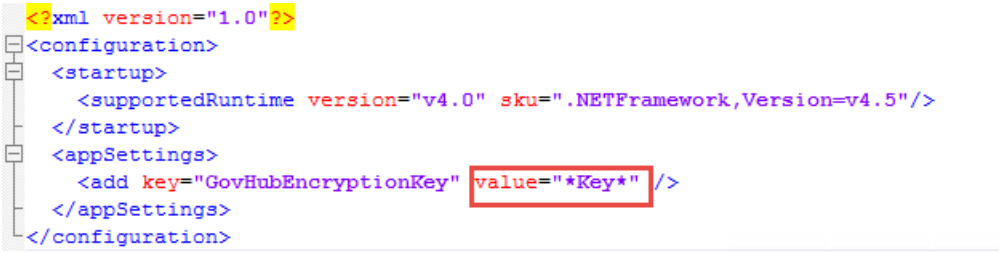
Note: To execute the ERP-Link Modeler, the following must be configured to support connecting to an SAP environment:

- ERP-Link Connector Service. Refer to the *ERP-Link V5.4.0 Installation and Administration Guide* for details on installing and configuring this product.

Configuration

Once the installation is complete, the ERP-Link Modeler will require some simple configuration to allow for communication with the Gimmal Connection Service. The ERP-Link configuration includes a key for authorizing access to use the Connection Service. Information about creation and management of the key can be found in the ERP-Link Administration Guide. The key that is used will need to be loaded into the Modeler configuration file (ERPLink.iNetModeler.exe.config). This file is typically stored in the \ProgramFiles\Gimmal\ERP-Link\Modeler file path.

The value specified in the ERP-Link configuration file needs to be set in the "value" as highlighted in the image below.



Any changes to the key in the ERP-Link configuration must be synchronized with the Modeler config file for it to connect properly.

Using the Modeler

You can perform the following tasks with the ERP-Link Modeler.

Selecting the Connection to SAP System

You must establish a connection to an existing SAP System using the Gimmal Connection Service. Please refer to the *ERP-Link V5.4.0 Installation and Administration Guide* for instructions to set up the connection.

1. To launch the Connection Browser, go to the **Start** menu under Windows and select **ERP-Link Modeler**.

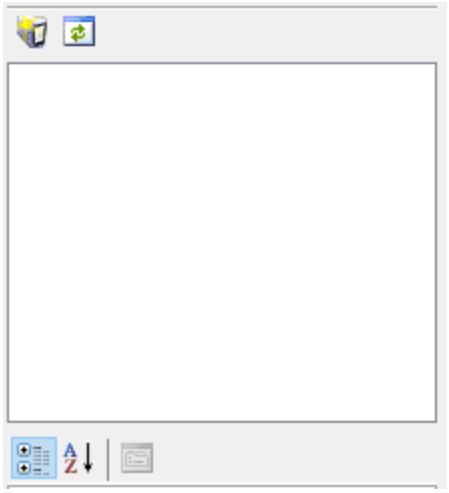


Figure 1 Connection Browser tool window

2. Browse the SAP systems by clicking the **Add SAP System** button in the toolbar.
3. Enter appropriate configuration information to establish a connection to an existing SAP system using the Gimmal Connection Service.

Browsing SAP Business Objects and BAPIs

When an SAP system has been configured properly, it appears as a tree node in the Connection Browser window. The tree node can be expanded to show its contents.

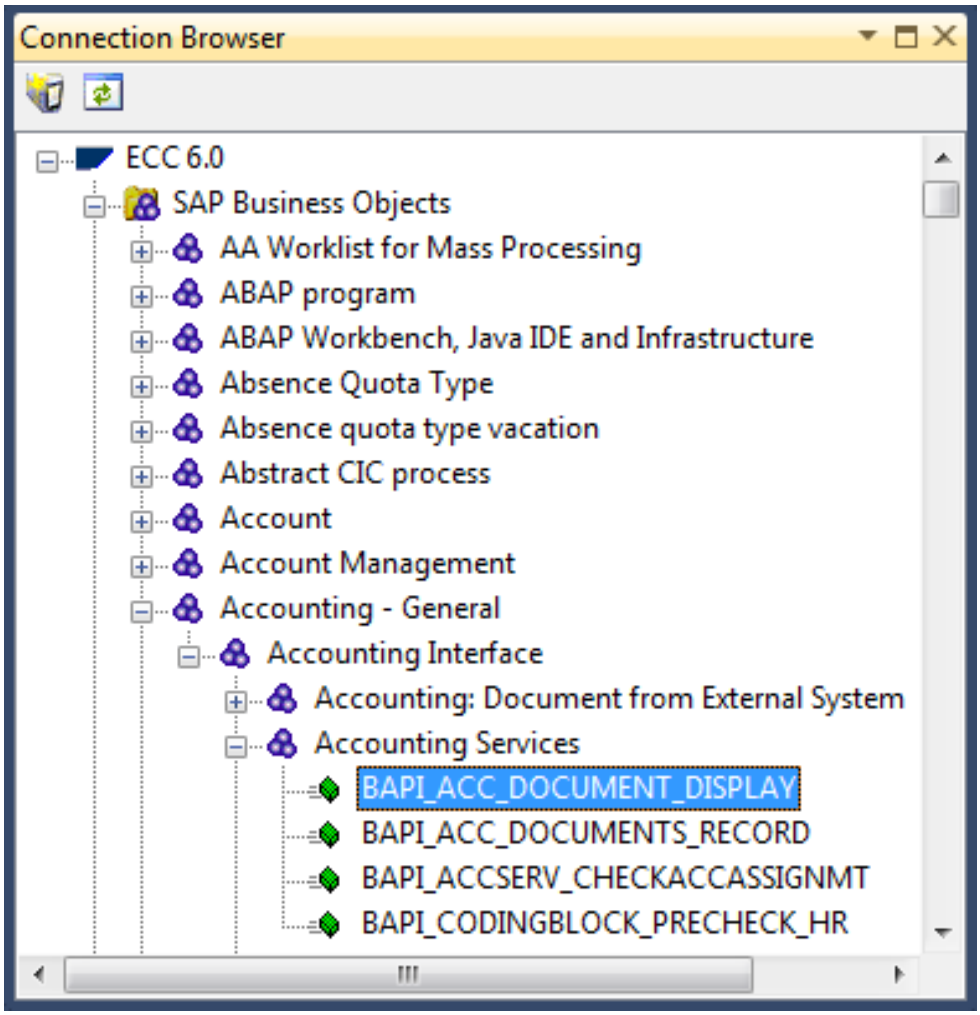


Figure 2 SAP Business Objects and BAPIs in the Connection Browser Dialog Box

An SAP system provides remotely callable services known as *BAPIs*. The BAPIs are grouped into SAP business objects based on their functional area. The BAPIs of a particular SAP system can be browsed by expanding the **SAP business objects** node of an SAP system in the Connection Browser.

Creating Remote Function Call Filters

If you already know part of, or the entire, name of a BAPI or a Remote Function Call (RFC) module, you can create a **filter.**

1. Right-click on the SAP system in the **Connection Browser** and choose **Add RFC Filter**.
2. Enter a descriptive name for the filter and the name of the group and function you are interested in searching for. Filter searches are always case-insensitive (that is, specifying *a* will match both *a* and *A*).

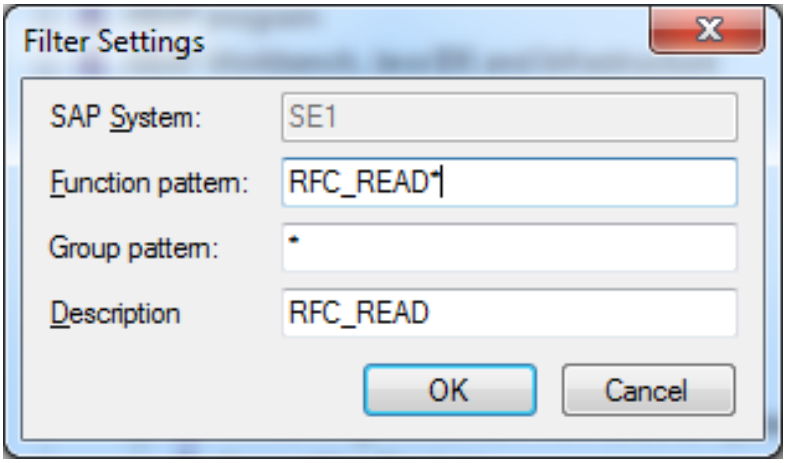


Figure 3 Add RFC Filter Settings Dialog Box

Note
Wildcards can be used in the search. The asterisk ('*') character is used to indicate *match everything*. For instance, entering the string Z_Test* in the Functions field would match the RFC modules Z_TEST_FUNCTION and Z_TEST_READ_TABLE– but not Z_GET_USER_DETAILS.

After either browsing the SAP business objects or running a filter, you will obtain a list of remotely callable functions, consisting of all BAPIs and RFC modules that match the filter criteria. Their use will be explained in the next section.

Browsing Tables

The Connection Browser provides tools to browse the tables on an SAP system.

1. Display a **Browsing Tables** dialog box by right-clicking on an SAP system in the Connection Browser.

2. Select **Browse Tables**. A dialog box displays to specify search criteria to browse the relational tables of that SAP system.

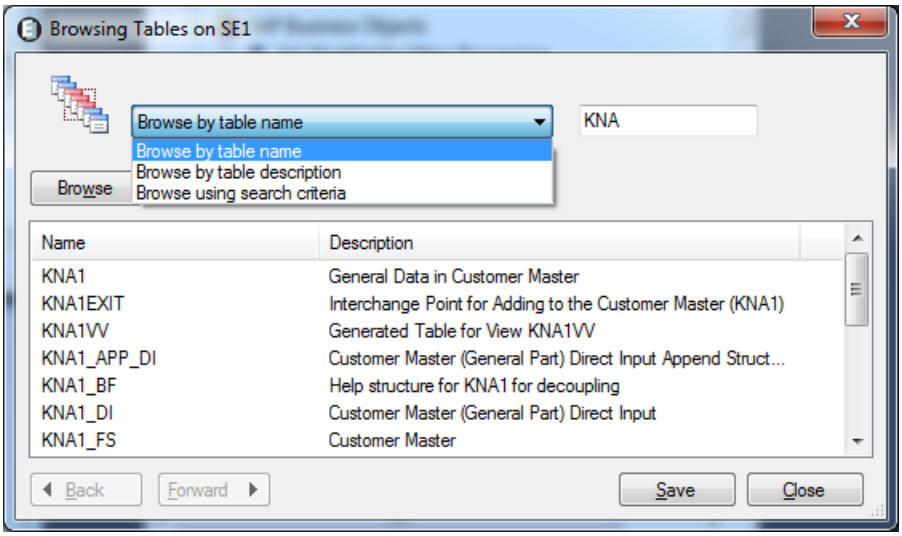


Figure 4 Browsing Tables Dialog Box

3. Enter the criteria and click the **Browse** button. Lengthy browse results can be paged through with the **Back** and **Forward** buttons as appropriate.
4. Save frequently performed table browses by clicking the **Save** button. The saved **Browse by table name** appears under the SAP system in the Connection Browser.

Searching with the Table Browser

Here are some tips for searching with the table browser:

- **Browse by table name:** Searches by the prefix of the names of SAP tables to display a list of those tables. For example, entering *KNA* results in the list of tables whose names begin with *KNA*.
- **Browse by table description:** Searches tables by their descriptions. For example, entering *Customer* will result in a list of tables whose descriptions contain the string *Customer*.
- **Browse using search criteria:** Displays an **Options** button. Click it to enter SAP SQL search criteria.

Viewing Tables

After you find the SAP tables that you are interested in, you can view the contents of a particular SAP table by double-clicking on it. A **Viewing table** dialog box displays.

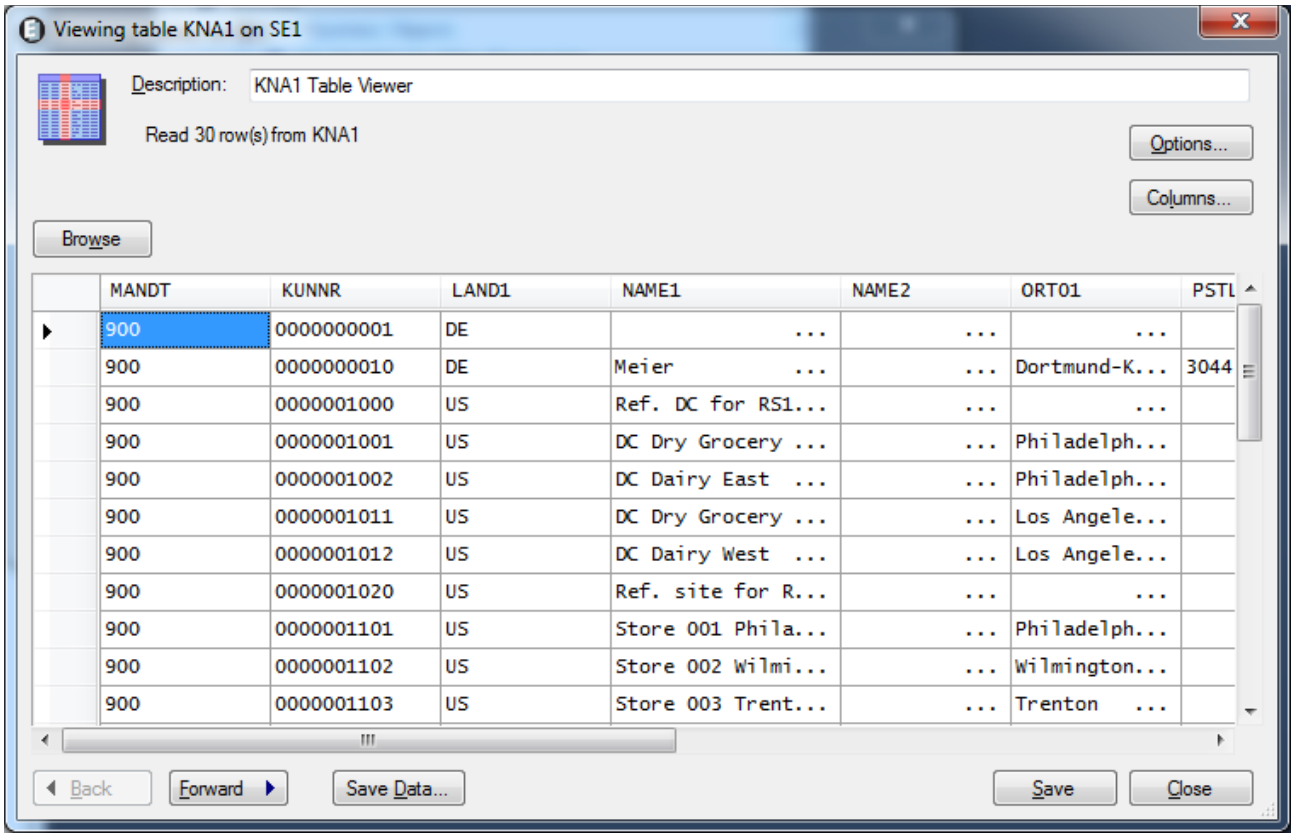


Figure 5 Table Viewer Example

- Use the **Back**, **Forward**, and **Browse** buttons to navigate through the table.
- Click the **Save** button to save the table view under the SAP System in the Connection Browser.
- Click the **Save Data** button to save the *entire* contents of the viewed table into a user-specified .xml file. This .xml file can then be imported into any application that can handle .xml files.

Note
Saving large tables might take a long time.

- Click the **Columns** button to display the **Select Table Columns** dialog box. This dialog lets the user choose the columns that are of interest.

Note
Due to technical limitations, the total width of all table rows retrieved by the Table Viewer can at most be 512 bytes wide. If this limit is exceeded, the Table Viewer will display the **Table View Column Selection** dialog box and prompt you to select columns with a total column width of 512 bytes or less.

- Click the **Options** button to display a dialog box to enter SAP SQL statements to limit the result set of the search.

Browsing SAP Queries

The Connection Browser displays the Queries available on a particular SAP system under the **SAP Queries** node.

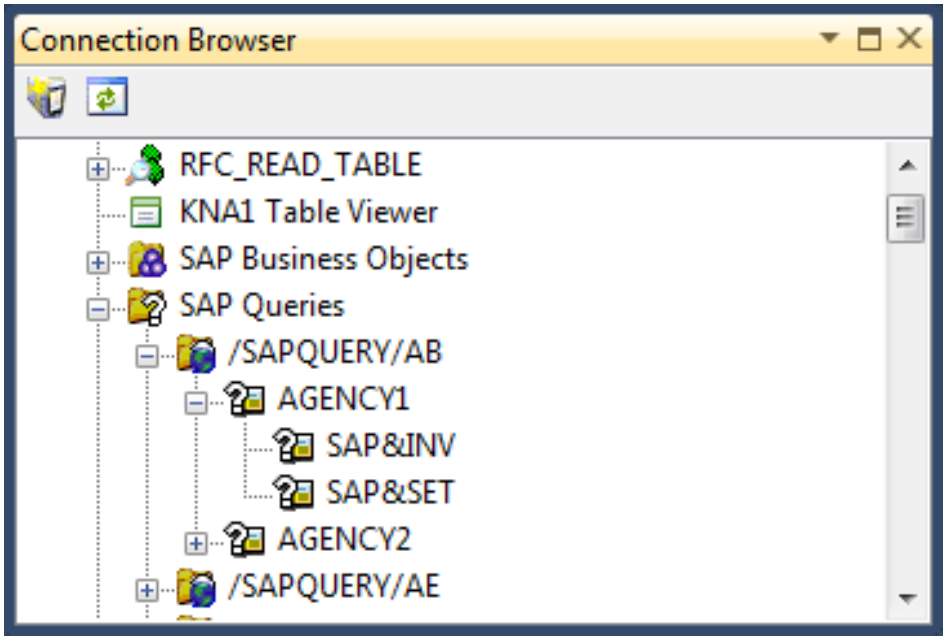


Figure 6 Browsing SAP Queries

SAP queries are grouped into *user groups*. User groups can be part of a global or local workspace; global user groups are indicated with an icon with a small blue globe. Each user group node contains one or more SAP *queries*, which in turn could contain one or more *variants*.

You can test a query or a variant from ERP-Link Modeler:

1. Right-click on the query or variant and select **Test Query** from the context menu.
2. Enter an optional number of rows.
3. Click the **Execute** button to invoke the selected SAP Query and display the returned data in the dialog.

Browsing SAP Report Programs

The Connection Browser can display SAP Report Programs.

Expand the **SAP Reports** node to display the list of available SAP Report Programs.

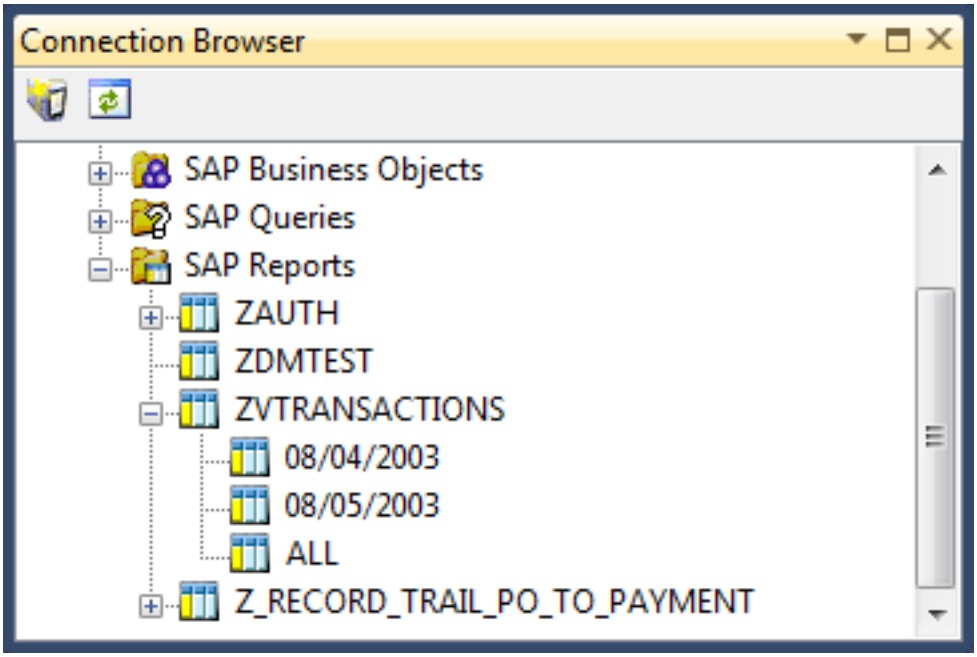


Figure 7 List of Available SAP Report Programs

Unlike queries, there are no User Groups or workspaces. SAP Report programs do, however, support variants in the same way that SAP queries do.

Using the ERP-Link Business Object Designer

You can use the ERP-Link Business Object Designer to design ERP-Link business objects intended to be used to generate a code snippet.

Creating ERP-Link Business Objects

You can add ERP-Link business objects to your Visual C# or Visual Basic.NET projects by modeling the desired objects in the ERP-Link Modeler and then generating a code file that can be imported into the project in Visual Studio.

1. Select **File** and then **New** from the menu in the ERP-Link modeler.
2. Select **ERP-Link Business Object** from the template pane.

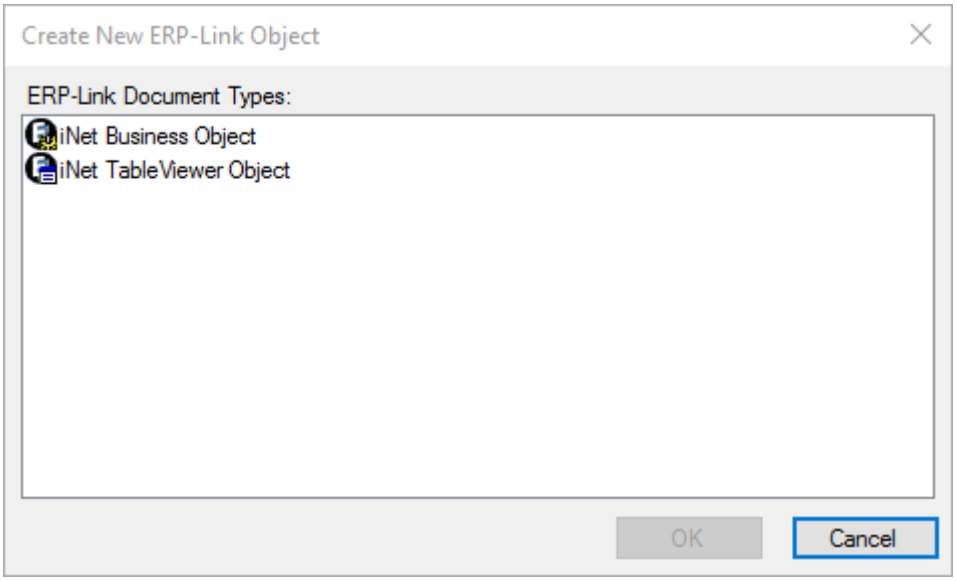


Figure 8 ERP-Link Template Chooser

3. Click **Cancel** in the **Create New ERP-Link Object** dialog box and a new, empty, ERP-Link business object displays in the designer window.

Note
Do not make modifications to the code in the proxy (*.ibx) file because the changes will instantly be lost whenever changes are made to the iNet business object.

The designer window consists of three areas.

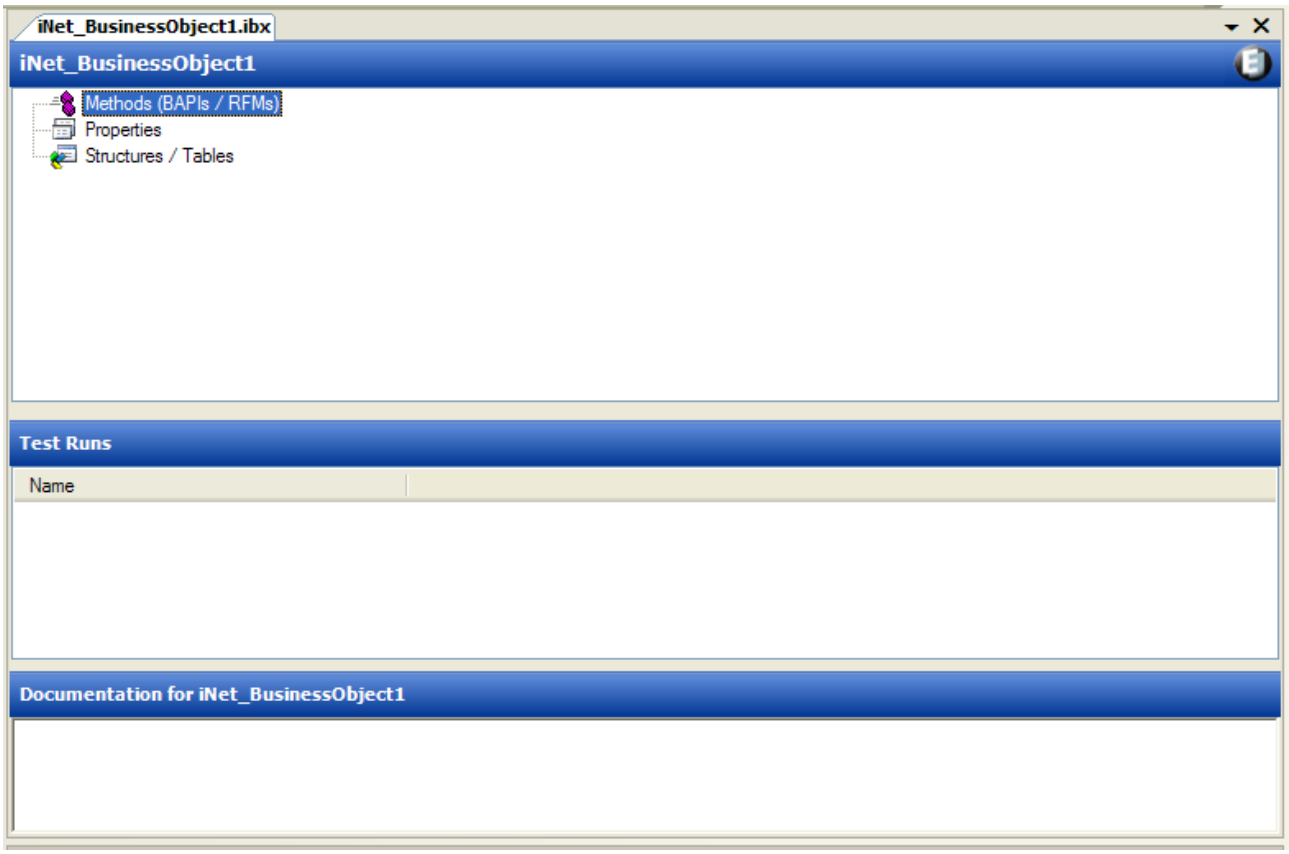


Figure 9 ERP-Link Business Object Designer

- The top area displays the **elements** of the business object being designed. A business object element is one of the following:
    - **Methods**, which correspond to BAPIs and/or RFC modules on an SAP system. Methods may have zero or more **parameters**
    - **Properties,** which can optionally be used to **bind** to parameters to improve performance
    - **Structures**, which are SAP structures and their fields
- The middle area, **Test Runs**, is where saved test runs from the Test Runner are presented (see Using the Test Runner). Launch saved test runs by double-clicking on them.
- The bottom area is the Documentation field. This field can be used to associate **Rich Text documentation** (including pictures and simple tables) with a particular method, parameter, property, structure, field, or column.

Adding BAPI/RFC Modules to an ERP-Link Business Object

To add a BAPI or RFC module to an ERP-Link business object, the BAPI/RFC module must first be visible in the Connection Browser, either by navigating the SAP business object hierarchy to the

appropriate BAPI or by using an appropriate filter. In the figure below, a filter to find all RFC modules whose names begin with RFC_READ_.

When the BAPI/RFC module is visible, click it and drag and drop it onto the Business Object Designer window. In our example, the RFC called RFC_READ_TABLE was dragged and dropped. The Business Object Designer will then look like the following figure:

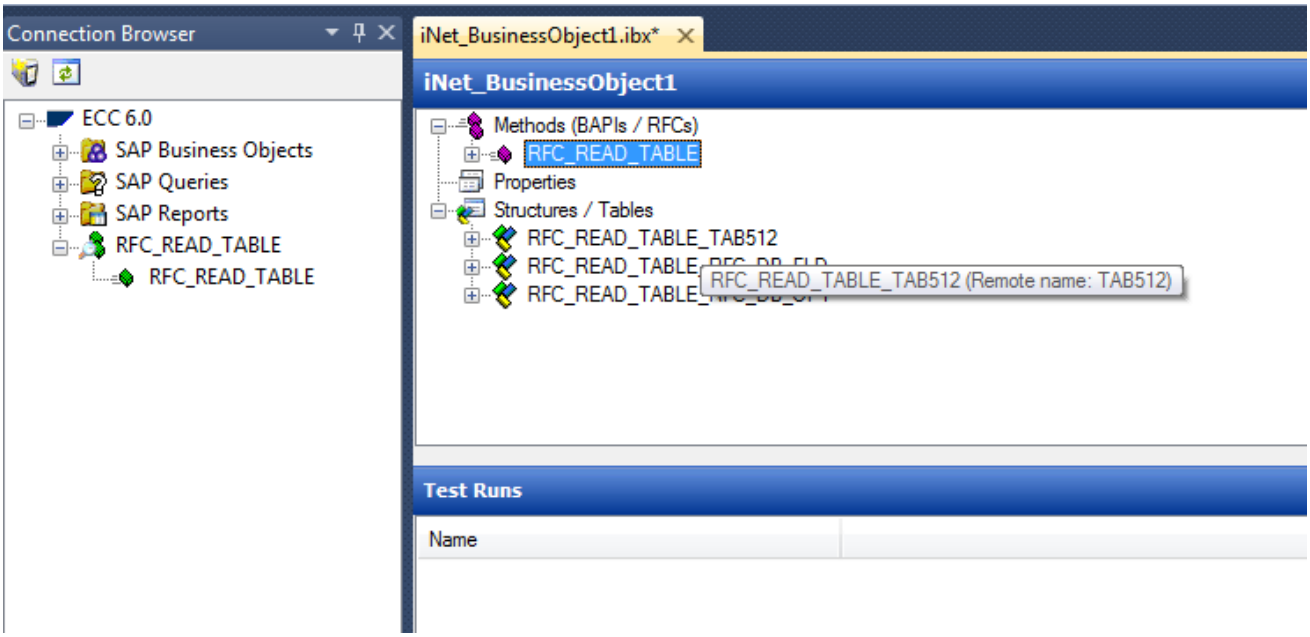


Figure 10 ERP-Link Business Object Designer Object with an Added Method

RFC_READ_TABLEis now the only method of the business object ERP-Link_BusinessObject1. You can add more methods by dragging and dropping them from the Connection Browser.

Note
It is not possible to add BAPIs/RFC modules from two or more separate SAP systems into the same ERP-Link business object, since it can only be targeted at exactly one SAP system. Instead, you can create a separate ERP-Link business object for each SAP system desired for communication. Alternatively, you can switch the targeted SAP system for the ERP-Link business object by changing its `SapSystem` property.

Deleting Methods from an ERP-Link Business Object

To delete a method from a business object in the Business Object Designer,

- Select the method and press the **Delete** key.
  or
- Right-click on the method and select the **Delete** menu item.

Renaming Methods and Other Elements

You can rename any method, method parameter, property, structure, table, structure field, or table column in the Business Object Designer.

- Select the element and press the **F2** key on the keyboard.
  or
- Right-clicking on the element and selecting **Rename**. The new name can then be typed in, followed by **Enter**.

Binding Method Parameters and Properties

By default, each BAPI/RFC module that is copied onto the ERP-Link Business Object Designer has all of its parameters replicated into the ERP-Link business object. These parameters can be renamed to suit the developer's needs.

However, in some cases, the parameters might always have the same value, or they might be optional and therefore should not appear to the .NET developer at all. Controlling these aspects of a parameter is called **binding** the parameter.

1. Locate the parameter in the Business Object Designer.

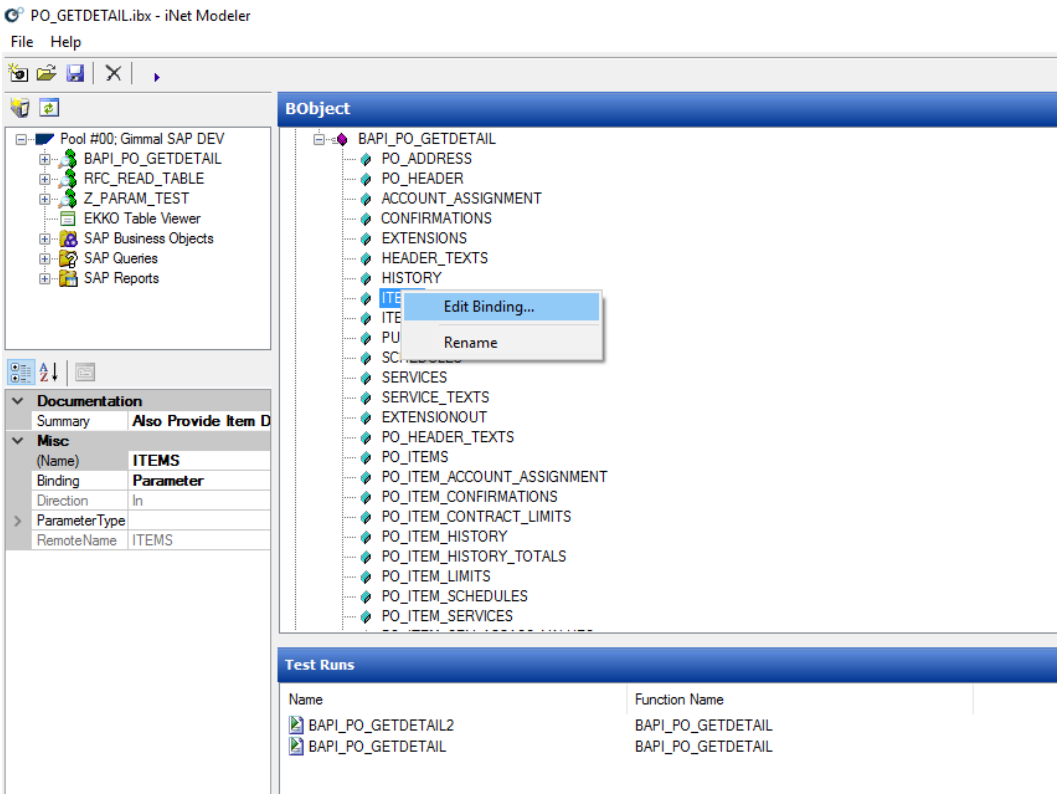


Figure 11 Parameter Binding Dialog Box

2. To bind a selected parameter, right-click on it and select **Edit Binding.** The **Editing Parameter** dialog box displays.

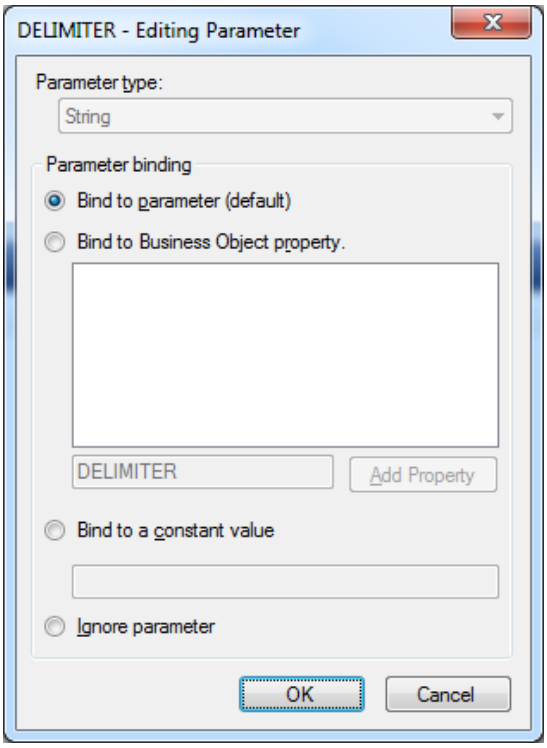


Figure 12 Editing Parameter Dialog Box

3. Edit the parameter following these guidelines.

   - In some situations, commonly used parameters can be bound to a .NET **property** on the generated proxy, for a slight performance gain. To bind a parameter to a property, choose **Bind to business object property**. A list of available, compatible properties will be displayed, one of which must be chosen for the binding to complete. New properties can be added to the business object by entering their name in the text field, then clicking **Add Property**.
   - If the proxy code should always pass a constant value to the BAPI/RFC module, select **Bind to a constant value** and then specify the desired constant value. The parameter will disappear from the generated proxy code because its value will always be constant.
   - If a BAPI/RFC module parameter is optional, and the requirement is to simplify the method interface by hiding that parameter from the .NET developer, select **Ignore parameter**. The parameter will disappear from the generated proxy code.

Using Structures and Tables

BAPI/RFC modules might have parameters that are structures or tables (**structured types**). Special functions are available for the management of structured types and their component fields or columns.

Sometimes, SAP structures or tables contain fields that are optional. Such fields can be made **visible** or **invisible** by right-clicking on them and toggling the **Visible** selection. Marking a field or column invisible causes it to disappear from the generated proxy source code.

When a method parameter is added to the ERP-Link Business Object Designer and that parameter has a structured type, a corresponding structured type is generated in the ERP-Link business object and the C# or VB.NET proxy. This generated structured type needs a name; by default, the name is generated by prefixing the original SAP name of the structure with the name of the method using the parameter. For example, if a parameter of the BAPI Z_TEST_00has the structured type TEST_STRUCT, then the resulting business object type name will be **Z**_TEST_00_TEST_STRUCT. These concatenations avoid unintentional conflicts between two methods using the same structure or table type.

If two BAPI/RFC method parameters are required to share the same structured type,

1. Select the parameter, right-click the selection, and select **Edit Binding** from the **Parameter Type** menu.
2. All compatible structured types are listed. Pick the one you want to associate with the parameter in question and then click **OK**.

Understanding Documentation and IntelliSense™

Whenever an ERP-Link business object element is selected, the Visual Studio property grid is updated to reflect the properties of that particular element. The **Summary** property, which accepts any text string, is common to most elements. This text string is generated in .xml-based comments that are then converted by Visual Studio into IntelliSense ToolTips. The .NET developer can edit the ToolTips after generating the snippet.

Most elements also allow the addition of free-form, rich text documentation in the **Documentation** pane at the bottom of the business object Designer.

- To add documentation, copy it to the clipboard from a rich text program such as Microsoft Word or Windows WordPad and paste it into the Documentation pane.

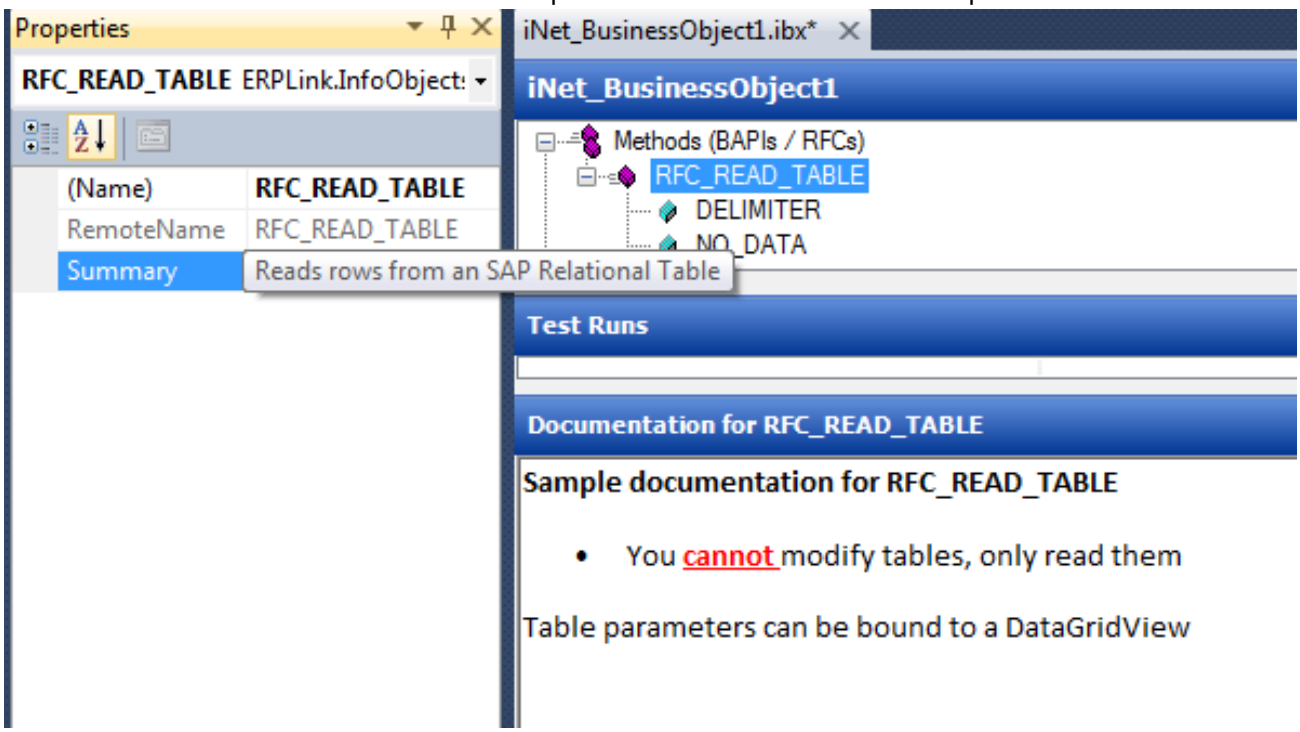


Figure 13 Addition of Documentation to a Business Object Element

Using the Test Runner

Sometimes it is convenient to be able to test a BAPI/RFC module without having to switch over to the SAP development environment. The ERP-Link Modeler provides functionality called the **Test Runner** that allows a user to select a method, specify parameter values, and execute the corresponding BAP/RFC module on the SAP system from inside of Visual Studio. To do this,

1. Select the desired method, right-click on it, and choose **Test Method**. The Test Runner window will appear.

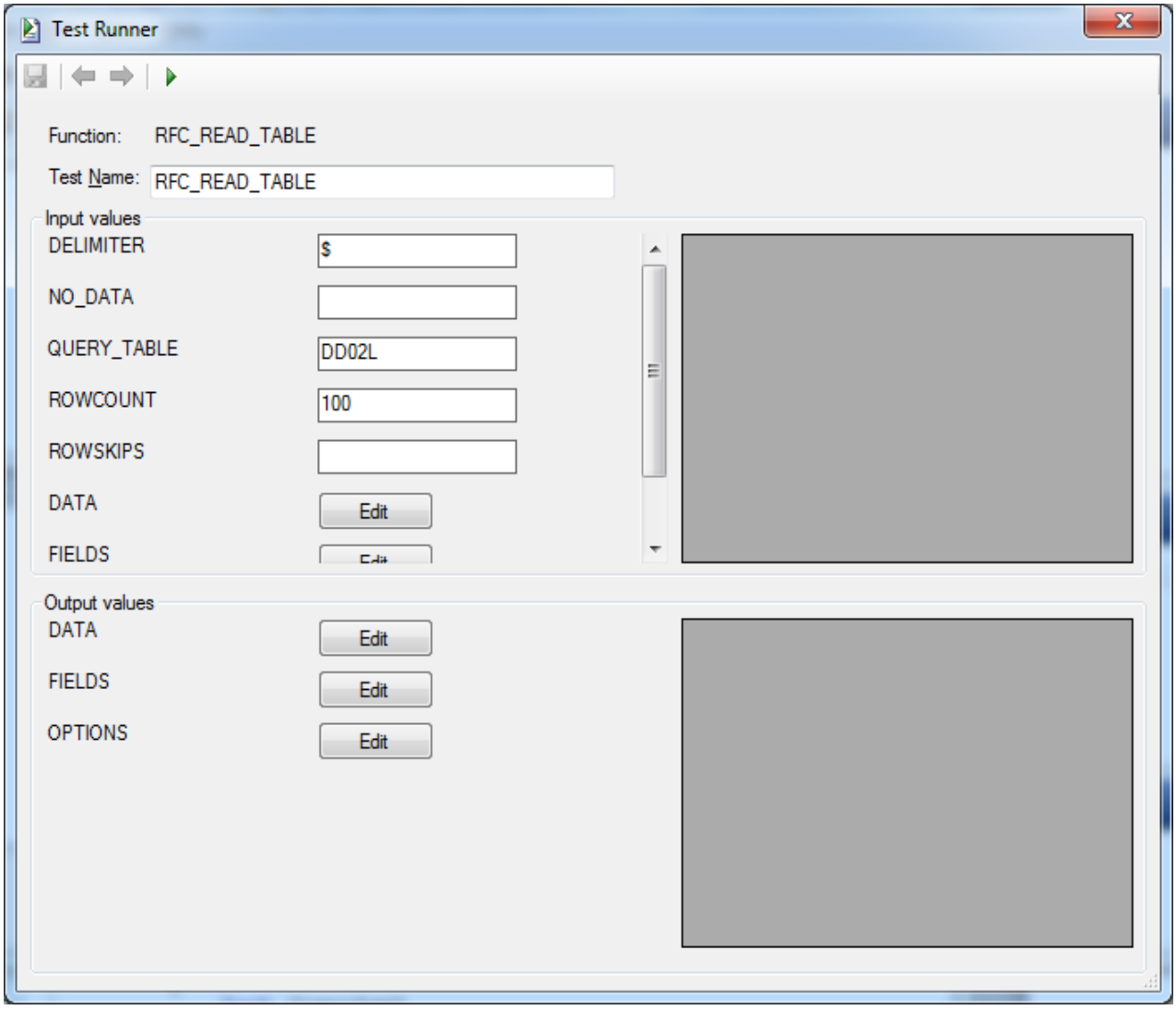


Figure 14 Test Runner Dialog Box

2. You can perform the following tasks with this window.

   - Enter simple parameter values by typing them into the appropriate text fields in the **Input values** pane.

- To enter data into structures or tables, click the corresponding **Edit** button. A data grid will appear, allowing data entry in a tabular manner.

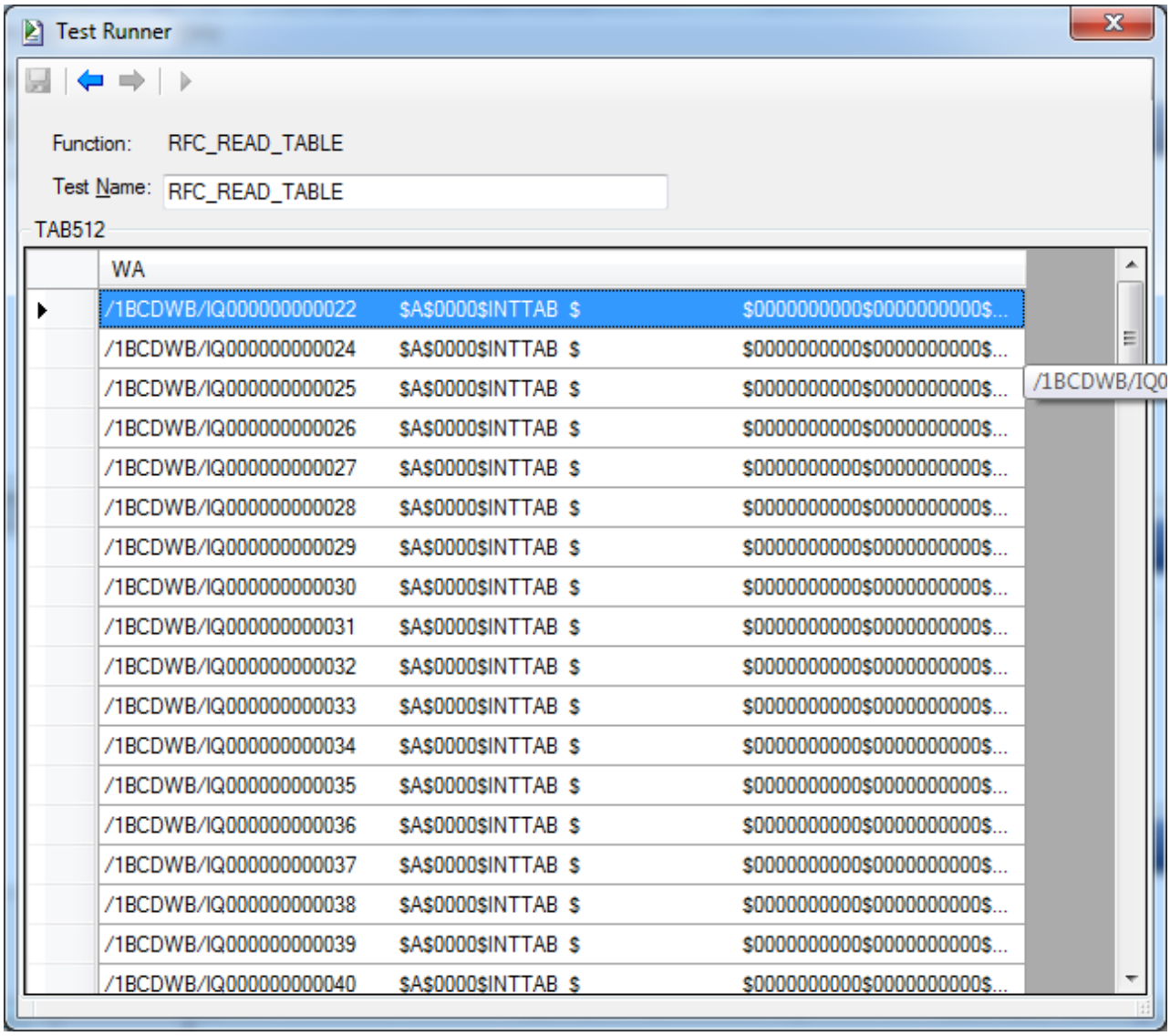


Figure 15 Test Runner Output Dialog Box

- You can save the values. Enter a descriptive name in the **Test Name** field and click the **Save** button. The test values are saved in a **test run** and are available for quick reuse later by double clicking in the **Test runs** panel of the business object Designer.
- When all input parameters have been entered, click the **Execute** button. A connection is established with the SAP system and the specified BAPI/RFC module is executed.
- Any returned values are visible in the **Output values** pane. To view structures or tables, click the corresponding **Edit** button.

Using the Table View Editor

The ERP-Link Modeler provides an information object designer called the *Table View Editor*. This editor lets you review and rename individual columns in a Table View. The resulting information object, an *ERP-Link TableView*, can be called from .NET code.

Creating a Table View

Add ERP-Link Table Views to your Visual C# or Visual Basic.NET projects by following these steps.

1. Select **File** and then **New**. The **Create New ERP-Link Object** dialog box appears.

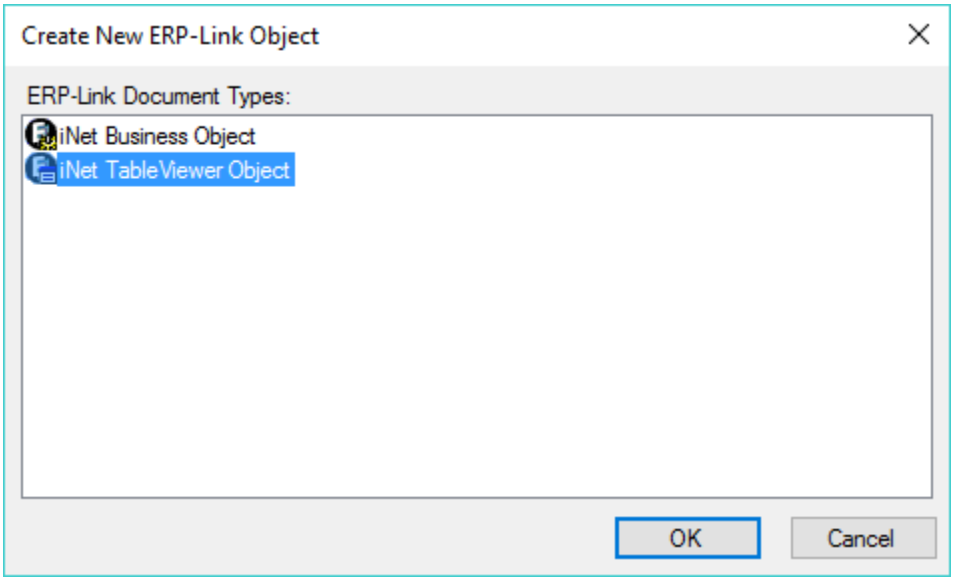


Figure 16 Adding a New Table View

2. Select the ERP-Link Table View Designer item in the Templates pane.
3. Click **OK**. A new empty ERP-Link Table View will be added as a file to your project and automatically opened in a Table View Editor.

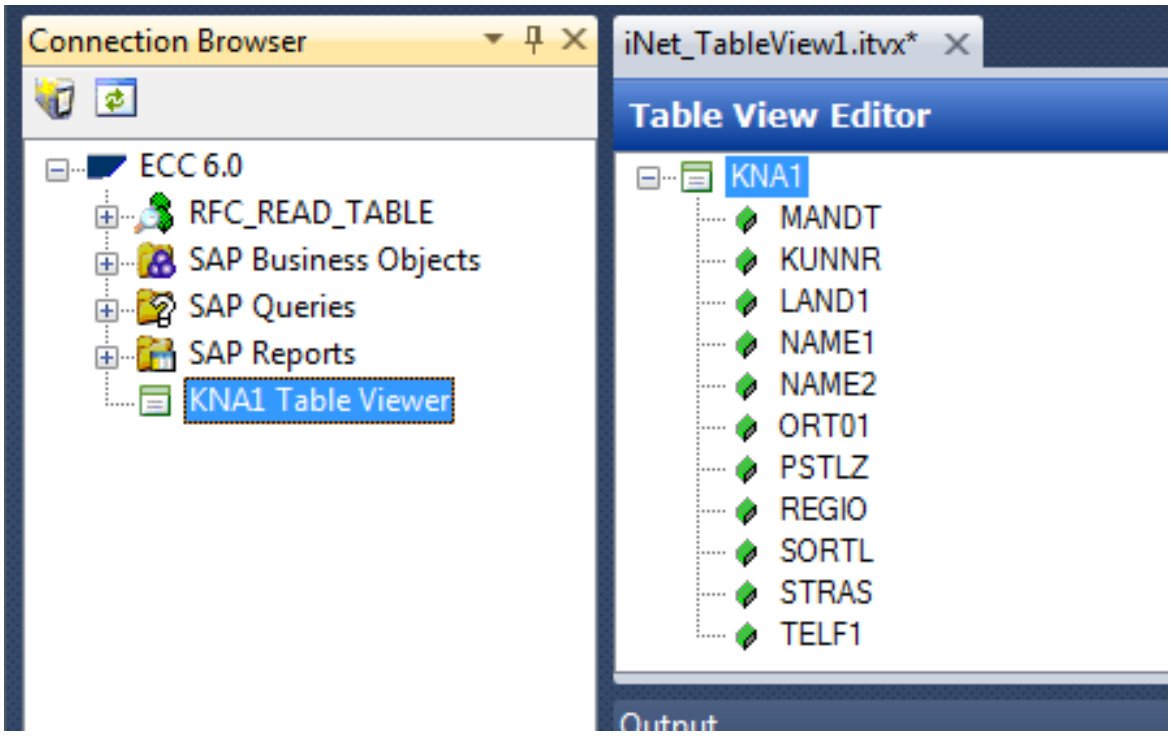


Figure 17 Table View Editor Window

Modifying a Table View

The Table View can be modified as follows:

- After a Table View is available in a designer, you can rename the individual columns of the Table View.
- You can change properties on the Table View itself in the Visual Studio **Properties** window.
- As you make changes in the Table View, the generated C# or Visual Basic.NET class changes to match those changes.

Using ERP-Link Information Objects in Your .NET Project

After you are satisfied with the designed ERP-Link information object, you can start using it to create a .NET assembly. This section will cover the programming aspects of using ERP-Link information objects in a .NET project.

Using the ERPLink.Runtime Assembly

All ERP-Link information objects require that you add a reference to a .NET assembly called *ERPLink.Runtime*, which implements the runtime connectivity to the SAP system. This assembly is installed by the ERP-Link Modeler.

In addition, redistributable **merge modules** for the 32-bit x86 and 64-bit AMD64 architectures, as well as a copy of the ERPLink.Runtime assembly, are installed in c:\Program Files\ERP-Link\Modeler.

ERPLink.Runtime is a thin abstraction layer that isolates the .NET programmer from implementation details of the different supported versions of ERP-Link SAP connections. By using this abstraction layer, .NET developers can use the same code against any version of services under any version of the .NET runtime without having to recompile their code.

A central class in ERPLink.Runtime is the ConnectionProviderService class. The ConnectTo (string connectionString) method of this class is the way in which a connection to an SAP system is established. The method has a return type of ISession. ISessions are used by ERP-Link information objects to communicate with SAP systems.

ERPLink.Runtime connection strings adhere to the following syntax:

PROVIDER=<prov>;SERVER=<server>;POOLID=<pool ID>

Where:

- The <prov> fragment identifies the version of the ERP-Link connection being used The ERP-Link Modeler 5.0 supports the ERP-Link Connection Service 3.1 and the Gimmal Connection Service 5.0. The <server> fragment identifies the name or IP address of the computer hosting the ERP-Link Connection Service server (such as localhostor 127.0.0.1).
- The <pool ID> fragment identifies the ERP-Link Connection Service connection pool to be used.

Example: the connection string

```
PROVIDER=iNet.CS3;SERVER=localhost;POOLID=42
```

specifies *use pool number 42 of the iNet.CS3 server on localhost to communicate with SAP.*

Performing Runtime Configuration of ERP-Link information Objects

At runtime; that is, after deployment, ERP-Link information objects can retrieve configuration information about which connection server and pool to use from your project's app.config or web.config file.

You can add the appropriate kind of configuration file and edit it to contain the following .xml elements:

```
    <configuration>

      <appSettings>

       <add key="BObject.iNet_BusinessObject1"
value="PROVIDER=iNet.CS3;SERVER=localhost;POOLID=01"/>

      </appSettings>

    </configuration>
```

The key should be the full .NET type name of the designed information object, including its namespace. Change the *localhost* and "01" to the appropriate values for the *runtime* environment of the project.

At runtime, the information object will locate its configuration from the appSettings section and use the retrieved connection string as a parameter to the ConnectionProviderService.ConnectTo() method described above.

Understanding Generated ERP-Link Business Object Proxy Source Code

This section discusses the high-level structure of the generated proxy code for a sample ERP-Link business object. The sample business object shown in Figure 18, below, has a single method targeted at the SAP standard RFC RFC_READ_TABLE. Notice that the method name and associated table types have been renamed. The ERP-Link business object will be used to read the table columns and the descriptions of these columns from an SAP system.

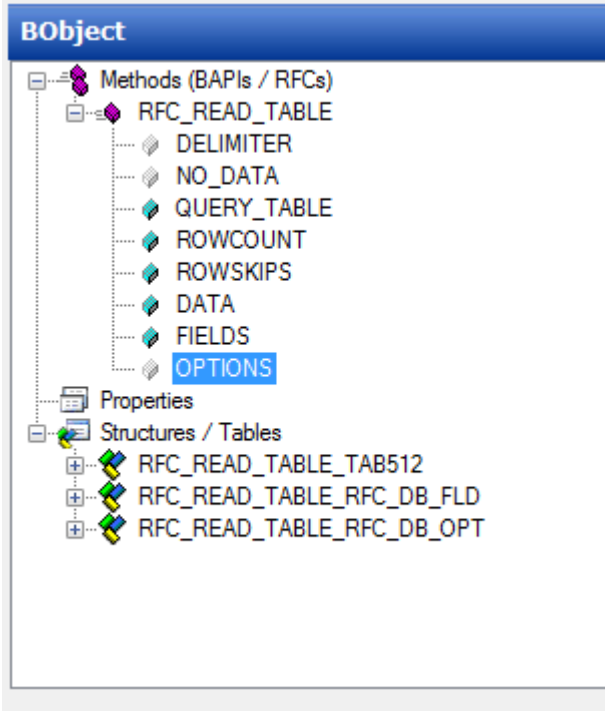


Figure 18 ERP-Link Business Object Method with Some Parameters Disabled

Notice also that three of the parameters of RFC_READ_TABLE, DELIMITER, NO_DATA, and OPTIONS are disabled. These parameters have been *bound to nothing*, or ignored, and will not generate corresponding .NET method parameters.

In the C# language, the proxy code that is generated will look like the following output (edited for readability and brevity):

```
public partial class iNet_BusinessObject1 : ProxyInfoObject {

    public iNet_BusinessObject1() {

    }

    public iNet_BusinessObject1(ERPLink.Runtime.SapCredentials credentials)

        : base(credentials)  {

    }

    public iNet_BusinessObject1(ERPLink.Runtime.iISession session)

        : base(session) {

    }

    /// <summary>

    /// Reads a selected number of rows from a table

    /// </summary>

    public virtual void ReadTable( string
        QUERY_TABLE,

        int ROWCOUNT, int
        ROWSKIPS,
```

```
            ref ERPLink.Runtime.RfcTable<TableData> DATA,  ref
            ERPLink.Runtime.RfcTable<TableFields> FIELDS)

        {

            ERPLink.Runtime.IRfcRequest request =
                this.Session.CreateRfcRequest("RFC_READ_TABLE");

            request.SetValue("QUERY_TABLE", QUERY_TABLE);
            request.SetValue("ROWCOUNT", ROWCOUNT);
            request.SetValue("ROWSKIPS", ROWSKIPS); request.SetValue("DATA",
            DATA);  request.SetValue("FIELDS", FIELDS);
            ERPLink.Runtime.IRfcResponse response = request.Execute(); DATA =
            response.GetTable<TableData>("DATA");

            FIELDS = response.GetTable<TableFields>("FIELDS");

        }

    }

    public class TableData : ERPLink.Runtime.RfcStructureBase {

        /* Edited: TableData implementation */

    }

    public class TableFields : ERPLink.Runtime.RfcStructureBase {

        /* Edited: TableFieldsimplementation */

    }

}
```

Look at the generated ReadTable() method. Notice that unlike the original SAP RFC_READ_TABLE, the parameters that were bound to nothing are not present, as desired.

Also note that in addition to the ERP-Link_BusinessObject1class, two auxiliary classes have been generated. Each of these classes corresponds to the structure of the elements of the DATAand FIELDS parameters of RFC_READ_TABLE.

The generated ERP-Link_BusinessObject1class has three constructors, each with a different set of parameters. This pattern of three constructors is present in all generated ERP-Link information objects. The choice of which constructor to use depends on the way an information object is associated with the ERP-Link Connection Service connection pool that links it to the SAP system.

Constructor with No Parameters

If the connection pool doesn't require authentication, use the first parameter-less constructor. That is, in the code, write:

```
iNet_InfoObject1 myObj = new iNet_InfoObject1();
```

When created this way, an ERP-Link information object will obtain all of its configuration information from the web.config or app.config file, and creates and configures an internal ISession object.

Constructor with SAP Credentials

If the connection pool does require SAP credentials, use the second constructor.

Note
The developer is responsible for obtaining and managing the SAP credentials in a secure fashion.

```
iNet_InfoObject1 myObj = new iNet_InfoObject1(new SapCredentials( cli,
    user, passwd, lang));
```

When created this way, an ERP-Link information object obtains its non-sensitive configuration information from the web.config or app.config file, and supplements it with the sensitive SAP credentials provided by the programmer. As in the parameterless case above, an internal ISession object is created.

Constructor with Configured ISession

If an ISession object needs to be shared between several information objects, or if the programmer wants complete control over the configuration of the ISession, use the third constructor. In this case, the information object will not take ownership of the session, but instead will assume that the caller is in charge of configuring and disposing of it properly:

```
iNet_InfoObject1 myObj = new iNet_InfoObject1(session);
```

Skinning the ERP-Link business object

Current best programming practices prescribe separating interface from implementation. This practice is especially useful when programming using unit testing frameworks and test-driven development (TDD). It is not practical to call a real SAP system using an ERP-Link business object in the context of a unit test because the target SAP system might be offline or it might throw unexpected exceptions that would interfere with the intended function of the unit test.

A common solution to this problem is to *skin* the object under test; that is, create a .NET interface out of all publicly accessible methods and properties, and then have the ERP-Link business object implement the interface:

```
public interface ITestable {

    void ReadTable(string tabName, ref RfcTable<RFC_DATA> data);

}


public partial class iNet_InfoObject1 : ProxyInfoObject, ITestable { public void
    ReadTable(string tableName, ref RfcTable<RFC_DATA> data) {
```

```
        // implementation elided for clarity
    }
}
```

Now, the unit test assembly can define a fake or mock class that also implements the ITestable interface and simulates the actual behavior of the real ERP-Link business object:

```
public partial class FakeInfoObject1 : ITestable {
    public void ReadTable(string tableName, ref RfcTable<RFC_DATA> data) {
        // Simulation code elided for clarity
    }
```

The unit test methods can now create instances of the fake class to simulate the ERP-Link business object.

To skin an ERP-Link business object:

1. Launch the ERP-Link business object designer and select the **Methods** node. The Visual Studio property grid will show the property InterfaceName.
2. Set this property to a non-blank value to generate a .NET interface with the same signature as the public members of the ERP-Link business object.

Disposing of ERP-Link Information Object Proxies

After use, every ERP-Link information object proxy must be disposed of properly to release ERP-Link Connection Service resources. Every ERP-Link information object proxy implements the .NET System.IDisposable interface. When the proxy object is no longer needed, its Dispose() method must be called to release the resources it holds onto. This applies to *all* ERP-Link information object proxies, regardless of which constructor was used to create one.

In C#, the using syntax construct makes this simple:

```
using (iNet_InfoObject1 myObj = new iNet_InfoObject1()) {
    myObj.CallSomeMethod();
}
```

The Dispose()method is called automatically when the usingscope is left, even if an exception is thrown by CallSomeMethod().

An alternative, more explicit (and verbose) way of achieving the same result is demonstrated below:

```
iNet_InfoObject1 myObj = null; try {
    myObj = new iNet_InfoObject1();
    myObj.CallSomeMethod();
} catch (Exception ex) {
    /* Handle the exception (or not) */
} finally {
```

```
    /* We arrive here even if an exception is thrown */ if
    (myObj != null)

        myObj.Dispose();

}
```

Using ERP-Link Information Objects

Assuming the configuration of an ERP-Link information object is correct, after the appropriate constructor is called, it is now ready to use. The mode of use depends on what kind of ERP-Link information object it is.

ERP-Link Business Object Sample Code Fragment

Example: Assume a developer is writing .NET code to display the details of a customer in the SAP system in an ASP.NET page or a Windows form. The developer creates an ERP-Link business object, and creates an RFC filter in the Connector Browser searching for BAPI_CUSTOMER_GETDETAIL. The resulting RFC is dragged onto the ERP-Link business object designer.

In a separate C# file, the developer writes the following method:

```
public void DisplayCustomerDetails(string customerID) {

    using (iNet_BusinessObject1 piNet_BusinessObject1 = new iNet_BusinessObject1()) {
        BAPI_CUSTOMER_GETDETAIL_BAPIKNA101 PE_ADDRESS; BAPI_CUSTOMER_GETDETAIL_BAPIRETURN
        RETURN;

        string CUSTOMERNO = "";

        piNet_BusinessObject1.BAPI_CUSTOMER_GETDETAIL(out PE_ADDRESS, out RETURN, customerID, "", "", "",
"");

        // Obtained a customer's info, display it.
        myForm.LabelTitle.Text = PE_ADDRESS.FORM_OF_AD;
        myForm.LabelFirstName.Text = PE_ADDRESS.FIRST_NAME;
        myForm.LabelSurname.Text = PE_ADDRESS.NAME;

        // ...etc

    }

}
```

In this sample, myForm is an ASP.NET Page or a Windows form on which user interface labels are located.

Assume a developer would like to use some data from a custom SAP table together with some ADO.NET data. The developer creates an ERP-Link Table View and then creates a Table Viewer on the (fictitious) SAP Table Z_TEST_TABLEin the Connector Browser, drags the Table Viewer onto the Table View Editor. He gives the ERP-Link Table View the class name TestTableView.

In a separate C# file, the developer writes the following method.

```
public System.Data.DataTable ReadTestTableData() {
    System.Data.DataTable table = new System.Data.DataTable();
    TestTableView tableView = new TestTableView();

    try {
```

```
            tableView.Fill(table);
        }
        finally {
            tableView.Dispose();
        }
        return table;
    }
```

The method calls the Fill()method of the ERP-Link Table View, which creates a standard ADO.NET DataTableand fills it with data from the SAP Table. The DataTable is then returned to the caller, which uses the data in the DataTable.

Generating ERP-Link Code Snippets

ERP-Link Modeler supports the quick generation of small code segments, known as ERP-Link Code Snippets that assist .NET programmers in writing code that calls remote SAP objects.

Business Object Code Snippets

Because some BAPI/RFC modules take a large number of input parameters, it can be quite tedious to write a call to an ERP-Link business object method from scratch. This is especially true if a large number of structures or tables are being used as parameters.

For convenience, the ERP-Link Modeler provides programmer assistance with the **ERP-Link Code Snippet** function. Select the desired ERP-Link business object method, right-click and select **Copy ERP-Link Snippet**. This will place some generated code on the clipboard. Now, switch to the program calling the ERP-Link business object and paste the contents of the clipboard. The results will be similar to the following:

```
#region ReadTable method call

iNet_BusinessObject1 piNet_BusinessObject1 = new iNet_BusinessObject1(); string
QUERY_TABLE = "";

int ROWCOUNT = 0; int
ROWSKIPS = 0;

TableData DATA = new TableData(); TableFields
FIELDS = new TableFields();

piNet_BusinessObject1.ReadTable(QUERY_TABLE, ROWCOUNT, ROWSKIPS, ref DATA, ref FIELDS);

#endregion
```

All scalar parameters are given default values, and all necessary structure or table parameters are created. The code snippet can now easily be modified to change the actual parameter values as appropriate.

Table View Code Snippets

Pregenerated calls to ERP-Link Table Views can also be obtained in a similar fashion by selecting an ERP-Link Table View object in the Table View Editor, right-clicking on the selection, and selecting **Copy ERP-Link Snippet**. This again places some generated code on the clipboard. When pasted into a C# file, an ERP-Link Table View Code Snippet will look like similar to the following:

```
#region Reading from SAP Table KNA1

System.Data.DataTable table = new System.Data.DataTable();
HRLogic.KNA1 tableView = new HRLogic.KNA1();

try {

    tableView.Fill(table);

}

finally {

    tableView.Dispose();
```

```
}

#endregion
```

SAP Query and SAP Report Code Snippets

Code snippets SAP Query and SAP Report information objects can also be obtained in a similar fashion. To add code to call a Query or Report program, first select the Query or Report in question in the Connection Browser, then right-click and select the **Copy ERP-Link Snippet** menu. Then, switch to a C# or Visual Basic.NET source file and paste the contents of the clipboard. In a C# source file, the result will look something like the following:

```
#region Execute SAP report ZEXTRACT_BALANCE
ERPLink.RFCConnector.ISession session =

    ERPLink.Runtime.ConnectionProviderService.ConnectTo(
        "PROVIDER=iNet.CS3;SERVER=?;POOLID=?");

ERPLink.InfoObjects.Runtime.ReportInfoObject report =
    ERPLink.InfoObjects.Runtime.ReportInfoObject(session);

report.ReportName = "ZEXTRACT_BALANCE";
report.Variant = "TEST1";

System.Data.IDataReader reader = report.ExecuteReader();

#endregion
```

Note

The end result of the ExecuteReader() call is returned as a System.Data.IDataReader. The data returned from the Query or Report can thus be accessed row by row by the user's program. The IDataReader can also be bound to ASP.NET controls like System.Web.UI.DataGrid for immediate presentation.