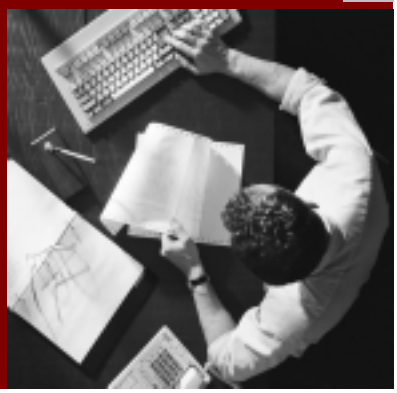


SAP® .NET Connector

Version 1.0

November 2002

SAP AG
Neurottstraße 16
69190 Walldorf
Germany
T +49/18 05/34 34 24
F +49/18 05/34 34 20
www.sap.com

Disclaimer

Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressively prohibited, as is any decompilation of these components.

Any Java™ Source Code delivered with this product is only to be used by SAP's Support Services and may not be modified or altered in any way.

Documentation in the SAP Service Marketplace

You can find this documentation at the following address:
http://service.sap.com/connectors

Typographic Conventions

Type Style	Represents
Example Text	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths and options.
	Cross-references to other documentation
Example text	Emphasized words or phrases in body text, titles of graphics and tables
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example, SELECT and INCLUDE.
Example text	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, source code as well as names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.
EXAMPLE TEXT	Keys on the keyboard, for example, function keys (such as F2) or the ENTER key.

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

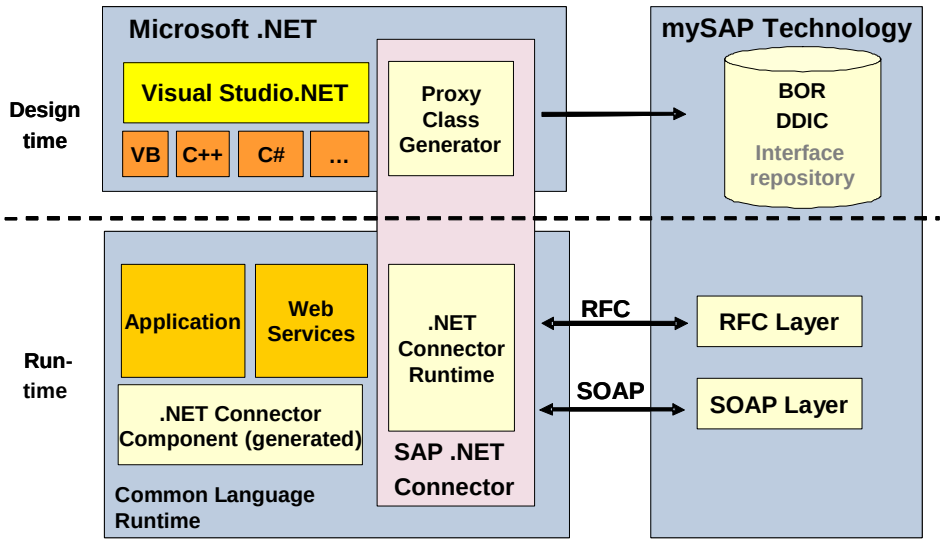
Contents

SAP .NET Connector

Part I SAP .NET Connector Overview

The SAP .NET Connector is a programming environment inside of Visual Studio .NET that enables communication between the Microsoft .NET platform and SAP Systems. It supports SAP Remote Function Calls (RFC) and Web services, and allows you to write various applications, for example, Web form, Windows form, and console applications within Microsoft Visual Studio .NET. You can use all Common Language Runtime (CLR) programming languages such as Visual Basic .NET, C#, or Managed C++.

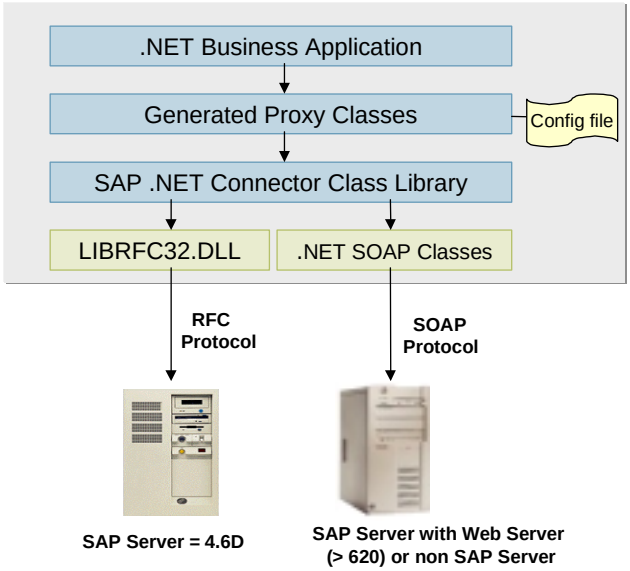
This documentation presents an overview of some key features of SAP .NET Connector and its architecture.



The connector is composed of several parts. First, there is extensive integration with Visual Studio .NET including a wizard for generating SAP proxies from the SAP data dictionary directly or from a WSDL file contained in the SAP IFR or a standard WSDL file. Within Visual Studio the connector includes several designer components to make developing SAP applications easier including destination components and an SAP Table component. SAP tables are a very important data type because they usually contain the results of the RFC call (for example a table of customer address data or sales orders). The SAP table component allows SAP table parameters to be data bound to most .NET data aware controls. Within Visual Studio you have graphical interfaces for the SAP proxies and you can easily customize your solution in a familiar way. This allows you to easily understand and work with the objects involved in interfacing with your SAP system and Microsoft .NET.

The Visual Studio developers can work in their choice of programming language (for example Visual Basic, C++, C#) to interact with the SAP proxies. The SAP proxies themselves are generated in C#. The SAP.NET connector provides a custom tool in Visual Studio so that SAP proxies can be automatically updated and customized instead of having to make these changes manually or having to rerun the proxy generation wizard.

Runtime Architecture



At runtime, the SAP proxies communicate with the SAP system by either the SAP RFC protocol (`librfc32.dll`) or via SOAP. SAP systems up to release 46D do not have SOAP support while SAP systems starting from 6.20 can use either SOAP or RFC. Non-SAP systems can be connected using SOAP. The SAP client solution is derived from the Microsoft `SoapHttpClientProtocol` class. The `SoapHttpClientProtocol` class is part of the Microsoft .NET framework and specifies the class proxies derive from when using SOAP. This base class for SAP clients provides .NET developers with a familiar way to use SAP functionality.

Deployment of the connector is made easier by use of configuration files so that values need not be hard coded in the application but instead updated in XML files. In addition, the connector supports all SAP authentication options and can be used in any type of Visual Studio .NET solution such as web services, ASP.NET web application, Windows forms, NT Service and more.

1 Features

Using SAP .NET Connector and SAP .NET Proxy Wizard, you can:

- Easily write .NET Windows and Web form applications that access SAP remote enabled functions (RFC)
- Write client applications for the SAP server using either RFCs or HTTP/SOAP/XML (outside-in)
- Write RFC server applications that run in a .NET environment and can be implemented from within the SAP System (inside-out)

You can develop entirely within Microsoft Visual Studio .NET:

- Use the Proxy Wizard integrated in Microsoft Visual Studio .NET to generate proxy objects that are easy to use
- Use any common programming language that has full access to the Microsoft .NET Framework
- Use IntelliSense help in Microsoft Visual Studio .NET through strongly typed data models and method signatures
- Bind SAP tables and structures to Windows and Web form controls (DataBinding)
- Use security authentication methods such as Single Sign-on, Kerberos, and Microsoft Passport

2 Prerequisites

Development System	Deployment System
• Windows 2000 or Windows XP • Microsoft Visual Studio .NET • Java Runtime Environment (JRE) You can download JRE version 1.3 or later from http://java.sun.com/j2se/1.3/jre • `SAP.Net.Setup.msi` Execute this file.	• Windows 2000 or Windows XP • Microsoft .NET Framework Download this file from http://msdn.microsoft.com/netframework • `LIBRFC32.dll`, Release 6.20 or higher • `SAP.NET.Connector.dll`

3 Creating an ASP .NET Web Application Using the SAP .NET Connector

The following example shows how to create a .NET project using Microsoft Visual Studio .NET. In the example, a client application reads and displays customer data from an SAP System using a search value and then displays it in a data grid.

The example uses the function module `RFC_CUSTOMER_GET`, which requires that customer data exist in the target SAP System, for example, in IDES. Although it is possible to rename all development objects and generated proxy classes, default names are used in this example. This example is provided as part of the connector sample code (`DNCWebApp`).

Procedure

1. Open Microsoft Visual Studio .NET.
2. Create a new C# Web form project:

   Choose *New* → *New Project* → *Visual C# Projects* → *ASP .NET Web Application*.

   You can also create a project in any other common programming language for .NET, for example, in Visual Basic .NET. In this case, you must add the SAP .NET proxy classes as a separate project in the Microsoft Visual Studio .NET solution.
3. Rename the form `Webform1.aspx` to `Default.aspx`.
4. Add Web controls to your Web form.

   In our example, we add a *TextBox*, a Button and a *DataGrid* control.
5. Add proxy classes to connect the Web applications to your SAP server.
   a. In the *Solution Explorer*, right-click on your project.
   b. Choose *Add* → *Add new item*.
   c. Select *Web Project Items* → *SAP Connector Class* and choose *Open.*

      The *SAP .NET Connector Wizard* opens.
   d. Decide from where you want to generate the proxy classes.

      You can create proxies from:

      - Web Services Description Language (WSDL) files that originate in an SAP interface repository (IFR)
      - An SAP server
      - Standard WSDL files
   e. Select the client proxy object type and select beautify names option.
   f. Select the Remote Function Modules (RFM) you want to use in your proxy object.

      You can use search filters to look for the Remote Function Modules. In the example, enter the search argument *RFC_CUST** in *Name-Filter* and select *RFC_CUSTOMER_GET.*
   g. Add the modules to your proxy object and choose *Next*.

The proxy classes for the referenced table and structure types are automatically created and added to the project.

6. Build the solution with *Build → Build Solution.*
7. Create an *SAPLogin* page to support user name and password authentication
   a. In the *Solution Explorer*, right-click on your project.
   b. Choose *Add → Add New item.*
   c. Select *Web Project Items → SAP Login Form*

   Leave the name as `SAPLogin1.aspx`.
8. Set the system connection information in the destination object of the `SAPLogin1.aspx` page:
   a. In the *Solution Explorer* window find the item `SAPLogin1.aspx` and double-click on it to bring it up in the designer.
   b. Look for the component `destination1` on the bottom of the form.
   c. Click on the destination component and set the properties for connecting to your SAP system (for example `AppServerHost` and `SystemNumber`). The other properties like client, Password and username will be set from the login page.
9. Databind the data grid to `BRFCKNA1Table`:

   `BRFCKNA1Table` is the parameter of `RFC_CUSTOMER_GET` that contains the list of customers.
   a. Select *SAP Table* Wizard from the SAP proxy toolbox and Drag&Drop it to your working area. In the dialog box, select `BRFCKNA1Table`.
   b. Select the data grid, and under *Properties* change *DataSource* to `BRFCKNA1Table` using the drop down list.
   c. Customize the list of columns displayed on the data grid by modifying the *Columns* collection property.
10. On the `default.aspx` page, double-click the *Button* control you added earlier to create an event handler for the control.
11. Add the connect code to your project:
    a. Select *Connect code* from the SAP proxy toolbox.
    b. Drag&Drop it in the source code of your event handler.

       A fragment of sample code is then inserted. It connects to the SAP server using the authorization settings from the Proxy Wizard. Normally, you must change these settings.

The code should look something like this:

```
private void btnSearch_Click(object sender, System.EventArgs e)
            {
// Declare parameters here
SAPProxy1 proxy = new SAPProxy1();
try
{
proxy.Connection =
SAP.Connector.SAPLoginProvider.GetSAPConnection(this);
// Call methods here
```

```
proxy.Rfc_Customer_Get("", txtCust.Text, ref brfcknA1Table1);
// Now update Data Bindings. On WinForms this will be automatic, on
// WebForms call the following line
this.DataBind();
}
catch(Exception ex)
 {
// If SAPLoginProvider.GetSAPConnection(this) cannot get a connection,
// we might get an error.
// Normally this can be ignored as it will automatically will force a
// relogon.
}
}
```

12. Build and run the application.

    The browser window opens and you are redirected to your `SAPLogin1.aspx` login page.

13. Enter connection data (for example user, password and client).

    If you select *Save* this login information will be stored as an encrypted cookie on your computer and will provide an alternative single sign-on capability the next time you wish to access the site. If you do not select *Save*, the login information will still be saved in the ASP .NET session state but will be lost once the browser is closed.

14. Enter a search argument, for example **A*** in the *TextBox* field and choose *Search*.

    Your application connects to the SAP System and displays the requested data in the DataGrid.

4 SAP Client Applications

The SAP client allows your .NET code to execute SAP functions that are remote-enabled (RFC). Some typical uses for an SAP client application include:

- ASP .NET web application to access information on the SAP system (for example sales orders status)
- Windows form application to provide a customized and highly interactive user experience (for example to enter sales orders)
- Console application to access information from the SAP system as part of some NT batch processing
- Web service to provide a SOAP interface to your SAP system prior to release 6.20, which includes native SOAP interface.

In the SAP client solution, the SAP system is the server and the .NET application is the client that interacts with the RFC. When you generate an SAP client application, a WSDL file is added to your Visual Studio .NET project. This WSDL file in coordination with a custom tool in Visual Studio creates several C# classes needed to communicate with the SAP system via either RFC or SOAP. There is one class for the proxy itself, one for each export parameter and two for each Table parameter in your RFC.

The SAP RFC is called as a method of the proxy object. There can be one or more RFC per proxy. For example you could have a proxy with all customer-related RFCs in one library. The parameters for each RFC can be customized within the Visual Studio designer so that optional parameters can be removed, parameters can be renamed and default values provided. You can also customize SAP structures by renaming or removing fields.

Visual Studio developers can work with the SAP proxies in their choice of programming language. The proxies themselves are generated in C# so for projects written in other than C# you have to add a new project of type `SAP Connector Class` to the Visual Studio solution.

4.1 Project Types

We recommend using the *SAP Connector Class* project template for creating SAP .NET Connector projects as you can reuse the proxy code. It is also worth considering placing the SAP proxies in the global assembly cache if several applications are making use of them.

Alternatively you can add the SAP connector class directly to another type of project such as a Windows Form, ASP .NET web application or ASP.NET web service. With other project types you can generate the SAP proxy directly in the project or reference an existing *SAP Connector Library*. When you select an existing *SAP connector library* you have to verify that your project has a reference to the SAP .Connector library (`sap.connector.dll`).

4.2 RFC or SOAP

The choice of whether to use RFC or SOAP depends on which release of the SAP system is available (for example, releases before 6.20) and other issues such as whether the system is available on your intranet or outside the firewall. Depending on the connection string used to create the proxy (for example, if it begins with `http://`), SOAP will be used to connect to the SAP system otherwise RFC will be used to connect to the SAP system.

4.3 Connecting to the SAP System

The connection to SAP is managed within the proxy's connection object. You do not have to determine the status of the connection yourself as the connector manages this automatically.

Before an RFC can be executed, the connection must be opened. After the RFC has finished executing the connection should be closed.

For applications that have many concurrent users, the connector provides a connection pool object. It is possible to get the connection from the pool instead of creating one for each client. In this way connections are reused and performance is improved.

4.4 Authentication

The SAP .NET Connector supports all SAP authentication options including user name, password and various single sign-on options such as Kerberos, NTLM, X.509 certificates and SAP Logon tickets. In addition, the connector makes it easy to perform SAP authentication in your application with the `SAPDestination` and `SAPLogonDestination` classes and in ASP .NET applications with the `SAPLogin Form.`

We recommend you use the SAP logon classes rather than manually creating a connection string with the logon information. The SAP logon classes support getting logon information at runtime from a configuration file, SAPGUI or as another alternative, programmatically, for example from Microsoft Active Directory.

4.5 Asynchronous Methods

Client applications support asynchronous method invocations. The main benefit of this is that your SAP client application remains responsive even when the RFC call is taking some time.

Many areas of the .NET framework support asynchronous programming. SAP .NET Connector classes are built in C# and take advantage of many .NET features including asynchronous method invocations. Asynchronous programming techniques are important with SAP RFC calls as some calls can take a long time to complete. During this time, a single threaded client application seems to be unresponsive as the main thread of execution is waiting for the SAP method call to complete. Many RFC calls happen very quickly and the user may not notice, others may take some time and the application appears to be unresponsive.

With .NET Connector, we can take advantage of many features of the Microsoft Common Language Runtime (CLR) including support for easy asynchronous programming. This powerful feature is not available on any other SAP connector at this time. In all other connectors, BAPI and RFC calls are synchronous calls.

The Asynchronous method calls use the standard .NET delegate callback mechanism. It provides the programmer with a familiar and powerful way of performing asynchronous calls in the SAP Client application.

When we use asynchronous methods from the .NET Connector wizard, the proxy contains two additional methods for each RFC: `Begin<RFC Name>` and `End<RFC Name>` and three additional variables to manage the asynchronous result.

When `Begin<RFC Name>` is called, CLR queues the request and returns immediately to the caller. The target method will be on thread from the thread pool. The original thread is free to continue executing in parallel to the target method. If a callback has been specified on `Begin<RFC Name>`, it will be called when the target method returns. In the callback, the `End<RFC Name>` method is used to obtain the return value and the `in/out` parameters. If the callback was not specified on `Begin<RFC Name>`, then `End<RFC Name>` can be used on the original thread that submitted a request.

4.6 TRFC and QRFC Support

Transactional RFC (TRFC) guarantees that a function module is executed in the target system exactly once. Queued Transactional RFC (QRFC) is a type of TRFC that is executed only once and in a particular order.

Client applications can be used with normal synchronous RFC, TRFC and QRFC. Normal Remote Function Calls (RFCs) are synchronous and are not guaranteed to execute only once or to execute in any particular order. Normal RFC calls return some value to the calling application, for example a list of customers. Transactional Remote Function Calls (TRFCs) and Queued Remote Function Calls (QRFCs) do not return anything if the function was successfully added to the SAP system. If the function was not added correctly, they throw an exception.

In the connector there are separate method signatures generated for TRFC and QRFC versus the normal synchronous RFC method. In the case of TRFC and QRFC, an additional parameter called a transaction ID (`TID`) is used as a unique identifier within the SAP system and when adding the function module execution request to the SAP system. A TID is similar to a GUID. In fact, GUIDs can be mapped back and forth to SAP TID using helper functions in the RFC library.

TRFC or QRFC should be used when information is added only once to the SAP system (for example, when adding a sales order or submitting an IDOC). In the case where a client application needs information from the SAP system, for example a list of customers, it makes no sense to use TRFC as the SAP system will return nothing back to the application other than an exception should something go wrong.

Queued RFC enforces the order of execution of the functions in the SAP system. To use QRFC you must have the following:

- A name for the RFC queue to use on the SAP system. If the queue does not already exist it will be automatically created.
- A TID
- A queue index to determine the sequence. The queue index begins with zero.

4.7 Monitoring and Debugging

The connector includes several exception classes so that error handling is robust and natural for a Visual Studio developer. In addition to these exception classes, there is extensive debugging and tracing support built into the connector.

You can debug from your C# proxy directly into the SAP function module by setting the `ABAP_DEBUG` flag. Debugging through to the SAP system is useful when you are getting unexpected results back from the SAP system. To use the `ABAP_DEBUG` option you must have installed `SAPGUI` on your developer workstation. You cannot use the integrated ABAP debug option with web applications because they run under another Windows context that is invisible to the interactive user.

Detailed traces can be written using the tracing flag. Alternatively you can set the environment variables `RFC_TRACE` and `CPIC_TRACE` to have trace files written to your application directory.

During initial design and debugging it is often useful to run the SAP `RFC` function directly in the SAP system using transaction `SE37`. It is easier to isolate the problem once you are sure that input values are valid. In addition, you should reference the SAP function module documentation. The SAP data dictionary, which is integrated in the function editor, also gives you information on valid input values.

In the SAP system, there are automatic formatting functions that are not available in the connector, for example to add leading zeroes to a customer or invoice number. If the function is working in the SAP system but not in your proxy, it could be that the SAP system has applied one of these automatic-formatting routines but you did not.

The SAP system provides extensive tracing and monitoring capabilities inside of the system as well, for example within the area *Tools* → *Administration* → *Monitoring*.

4.8 IDOC

SAP Intermediate Documents (IDOCS) are EDI like documents that are asynchronous in nature. IDOCS are often used in sending business documents (for example sales orders) from your SAP system to a trading partner or other system. The actual TRFC call to submit the IDOC to SAP is performed synchronously and very quickly, but the actual business processing can happen at some later time defined in the SAP system. The outbound result (for example, sales order confirmation) can also happen at some later time. With RFC calls, the business processing is done immediately albeit on a different thread if we make use of the asynchronous methods described above. IDOCS offer additional queuing and retry capabilities.

SAP .NET Connector supports both submitting and receiving SAP IDOCs. The `SAPIDOCSender` class submits IDOCs and the `SAPIDOCReceiver` class can be used with a TRFC server to receive IDOCS.

To work with IDOCS you must use transactional RFC. In the `SAPIDOCSender` and `SAPIDOCReceiver` classes, SAP provides for you a TRFC client implementation that works with the appropriate function modules in SAP.

5 SAP .NET Server Applications

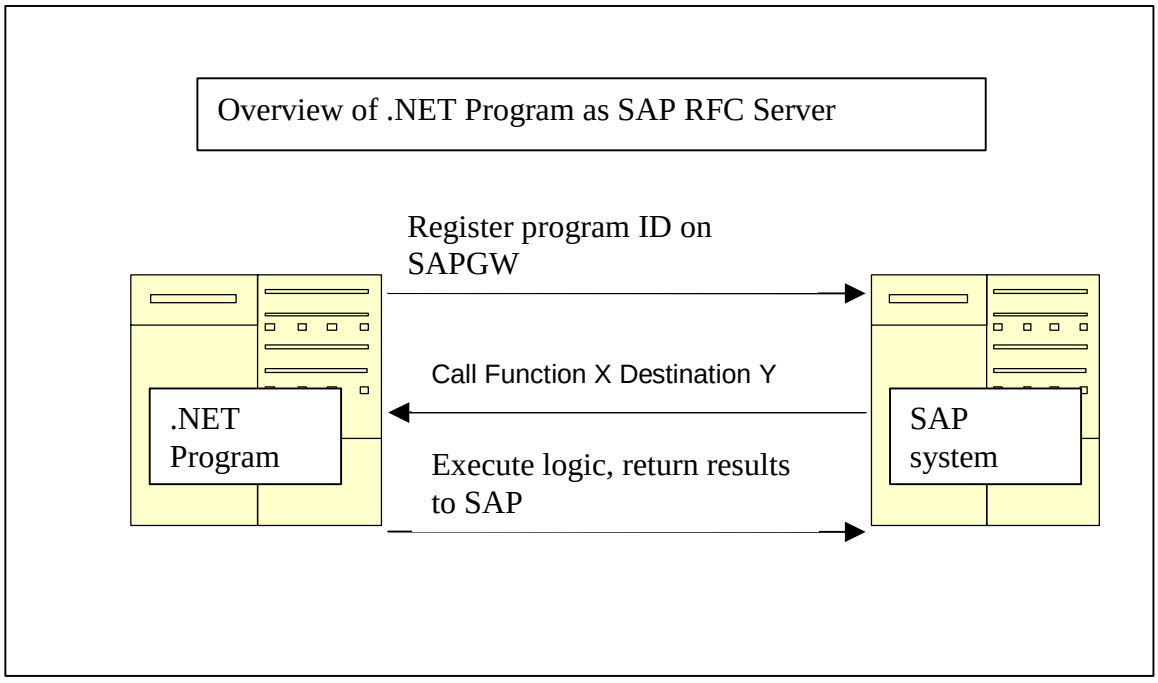
The RFC server allows your SAP system to execute .NET code as if the .NET code were another SAP system. The project type you choose for an SAP server project depends on your specific requirements but typically you choose between a console application, executable or Windows Service. With the server stub project you can use standard Visual Studio .NET debugging.

With SAP .NET Connector you can easily write RFC server programs in C#. It allows you to use functionality in .NET as easily as if it were in the SAP system.

You can use the SAP .NET Connector server stub for scenarios such as:

- Retrieving information from another system, for example additional customer or tax information necessary to process a sales order
- Getting information such as maps, stock prices, shipping information, flight information or weather from an external service to be used within an SAP report
- Sending emails from your SAP system
- Sending IDOCS from the SAP system to an external system

The following graphic shows the main processes of the .NET program and the SAP system. In this type of application, the SAP system is the client and the C# proxy is the server. All program logic is therefore done in the C# proxy and returned to the SAP system in an export, changing or table parameter.



An RFC server application allows you to use .NET functionality within your SAP system. RFC servers can be (normal) RFC, TRFC or QRFC servers.

5.1 Key Steps

When calling a .NET program from an SAP system, we distinguish the following key steps, as shown above:

- The .NET program must register itself on the SAP gateway host
- The SAP system and the .NET program must implement the same method interface, for example, the function module name and parameters from the SAP system.
- The SAP system initiates the call to the .NET program using the `CALL FUNCTION DESTINATION` keyword
- The .NET program must return the appropriate parameters to the SAP system

The SAP server code that is generated by the connector and provides the functionality described above as well as a default implementation. This allows the Visual Studio developer to focus on the implementation and not have to understand how RFC servers work in detail.

5.2 Authentication

Unlike client applications, no user name and password is required to register a server on the SAP gateway. Instead, the program ID, gateway host, gateway service and code page must be provided. Once the server is registered on the gateway inside the SAP system, it can be accessed from SAP programs. The computer where the RFC server is running must grant permissions to the SAP service user to use the necessary resources on that machine.

5.3 Monitoring and Debugging

After your program is running, use the SAP gateway monitor (transaction `SMGW`) to verify that your .NET program is registered. If the registration works properly, you can see on the *logged on clients* screen the `PROGID` and the host machine.

The SAP Gateway offers detailed tracing capabilities from within the SAP system. On your RFC server you can also enable tracing by setting the environment variable `RFC_TRACE` and `CPIC_TRACE`. The traces can be quite large so should only be used when there is a problem.

For TRFC servers you also have the TRFC monitor (`SM58`). For QRFC you have the SAP Queue monitor (transaction `SMQ2`) to monitor QRFC calls.

Just as the SAP client solution offers integrated Abap/4 debugging for .NET clients, you can set a breakpoint in your RFC server code, call the function from the SAP system and examine the values sent to you from the SAP system in debugging mode within Visual Studio .NET.

5.4 TRFC and QRFC

A TRFC server makes sense when have to send information only once from the SAP system to another application (for example, sending a purchase order). TRFC is required if you want to write an application to receive SAP IDOCS. On your TRFC server, you must manage a connection to a transactional store such as Microsoft SQL Server. You require a transaction store to ensure you can keep track of and manage all `RfcTID` sent to you from the SAP system so that you can create the TID and the function execution within a transaction. If the transaction fails at any point, the SAP system tries to resubmit it. By default, the system attempts to resubmit the transaction every 15 minutes up to a maximum of 30 attempts. However, you can configure the resubmission parameters individually for each RFC destination using transaction `SM59`. From the destination maintenance screen, choose *Destination → TRFC options.*

To call our TRFC server, the SAP system uses the syntax: `CALL FUNCTION xyz IN BACKGROUND TASK DESTINATION dest` and then issues a `COMMIT WORK`.

After the `COMMIT WORK`, the SAP system makes several additional calls related to TRFC against our .NET component. The first call is `CheckTID`. This method is responsible for checking whether we have already processed the TID. If we have processed the function, nothing further is done. If not, the next step is to execute the function itself (for example, function `xyz`) or queue the function for later execution. After successful completion of this function, we let the SAP system know that the transaction worked by calling `OnCommit`. Once the SAP system receives the commit from us it knows that everything worked properly and lets us know that we can clear the TID record from our database.

Although it is possible, we do not recommend writing a QRFC server with the connector. To write a QRFC server, you have to implement the SAP queuing mechanism. For purposes where you want queuing, we recommend to simply use *Microsoft Message Queue* in a TRFC server.

Part II SAP .NET Connector Programmers' Reference

1 Overview of Classes

Class	Explanation
`SAPClient`	Base class for all SAP client projects
`SAPServer`	RFC server class that allows you to make use of .NET functionality inside of your SAP Abap/4 programs
`SAPConnection`	Manages a connection to the SAP system. It is used by the `SAPClient` classes
`SAPConnectionPool`	Allows you to manage a pool of connection objects. This is important for applications where multiple users access your proxy
`Destination`	Base class for `Destination` objects. Holds login attributes as properties but does not contain logic to retrieve these properties as does `SAPLogonDestination` for example
`SAPLogonDestination`	Derived from Destination. In addition to holding login attributes this class can retrieve login information from the SAPGUI
`SAPLoginProvider`	User name/password login support via ASP .NET forms authentication. Also provides an alternative single sign-on capability
`RfcException`	Base exception class for SAP .NET Connector exceptions. Not raised by itself
`RfcAbapException`	Exception representing an Abap/4 exception raised by the SAP RFC Abap/4 code
`RfcCommunicationException`	Exception representing a communication failure of some type (for example, the SAP system is unreachable)
`RfcLogonException`	Exception representing a logon failure (for example, incorrect user name or password)
`RfcSystemException`	Exception representing a system error (for example, an SAP short dump has occurred)
`RfcQueueItem`	Used for QRFC calls to contain the SAP Queue information
`RfcTID`	Used for TRFC and QRFC calls. A TID is similar to a system guid

`SAPTable`	Very common data type used in RFC programming. This class can be data bound to most .NET data aware controls such as datagrids and listboxes. A example of an `SAPTable` might be a list of customers with name and address data for each row. `SAPTable` is a collection of `SAPStructure`
`SAPStructure`	Very common data type used in RFC programming. For example, a BAPI return code is an `SAPStructure`. An individual row in a table is a structure
`SAPIDocSender`	A class for submitting SAP intermediate documents (IDOCS) from text files
`SAPIDocReceiver`	A class for receiving SAP IDOCS from SAP.

2 SAP Client Programming

2.1 SAPClient Class

`SAPClient` is the base class for .NET applications wishing to use SAP functionality. This class is maintained by the `SAPConnectorGenerator` custom tool and the `SAPProxy wsdl` metadata file. It should not be necessary to manually update it. SAP client proxies are instances of this class.

For a list of members of this type see SAP Client Methods [Page 27].

```
[C#]
Public class SAPClient:
System.Web.Services.Protocols.SoapHttpClientProtocol
```

Remarks

In the SAP client solution, SAP is the server and the .NET application is the client that interacts with the RFC. When you generate an SAP client application, a WSDL file is added to your Visual Studio .NET project. This SAPWSDL file in coordination with a custom tool in Visual Studio creates several C# classes needed to communicate with the SAP system via either RFC or SOAP. There is one class for the proxy itself, one for each export parameter and two for each Table parameter in your RFC.

The SAP RFC is called as a method of the proxy object. There can be one or more RFC per proxy. For example you could have a proxy with all customer related RFCs in one library. The parameters for each RFC can be customized within the Visual Studio designer so that optional parameters can be removed, parameters can be renamed and default values provided. You can also customize SAP structures by renaming or removing fields.

Visual Studio developers can work with the SAP proxies in their choice of programming language. The proxies themselves are generated in C# so for projects written in other than C# you have to add a new project of type SAP `Connector class` to the Visual Studio solution.

2.2 SAPClient Proxy Generation

To add the SAP Proxy to your Visual Studio project you have the following options:

- Create a new project of type SAP `Connector Class`
- Add the proxy to another project type (for example `Winform`, `ASP .NET project`, `webservice`)

Procedure

1. In Visual Studio .NET choose *Project* → *Add New Item.*
2. Under templates select SAP `Connector Class`

   This brings up the connector wizard.

You can create proxies from one of these:

Proxies to Generate From	Explanation
`WSDL file from SAP IFR`	Web Services Description Language (WSDL) files that originate in an SAP interface repository (IFR). If you want to create proxies from an SAP IFR, you must have a URL for the WSDL file In the near future, the SAP IFR will be the central repository for SAP metadata including WSDL files
`SAP Server`	Generate the C# proxy by examining the metadata stored in the data dictionary of the SAP system. For most scenarios this is the preferred option
`Standard WSDL file`	You can use an existing WSDL file or create your own WSDL file and import it

3. If you generate proxies from an SAP server, enter the following information in the *Enter Logon information* screen:

Input	Explanation
`System`	SAP system name. If you have SAPGUI for Windows installed, you can select an entry from the drop down box
`Host`	Application server host name
`ID`	System ID (for example 0)
`Client`	SAP client number (for example, 000 or 800)
`User`	SAP user name
`Password`	Password for that user
`Object type (client proxy)`	The wizard creates all necessary code for a `SAPClient` project. The .NET application

Input	Explanation
	which uses SAP system functions is called a client proxy
`Object type (server stub)`	The wizard creates all necessary code for a SAPServer project. The SAP system which uses .NET functions is a server stub
`Beautify name`	SAP function names are by default upper case. This option makes the names more readable
`Create asynchronous methods`	This option is only for client applications. It allows you to make your SAP function call on a dedicated thread so that your application remains responsive should the call take some time

4. Select the Remote Function Modules (RFM) you want to use in your proxy object.

   You can use the following search filters to look for the function modules:

Option	Explanation
`Name-Filter`	A search string for finding the proper SAP function(s) by name
`Group-Filter`	SAP functions are organized by groups. If you know the group name you may use it here

After the code has been generated, you can see an SAP proxy WSDL file added to your project. This WSDL file in coordination with Visual Studio and the `SAPConnectorGenerator` custom tool allows you to manage the SAP proxies automatically.

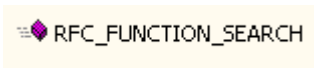
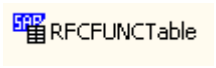
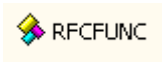
If you choose to generate the proxy from the SAP IFR you will be asked for the URL of the IFR Server.

If you choose to generate a proxy from a Standard WSDL file you will be asked for the path or URL to that WSDL file.

2.3 Customizing SAP Proxies

There is design time support for modifying and updating the generated proxies. To see the design view, simply double click on the generated SAPProxy WSDL file in the Visual Studio designer Solution Explorer.

The proxies should be customized from within Visual Studio designer. There is no need to rerun the proxy generation wizard unless the RFC signature has changed.

Type of Object	Example in Designer
`RFC functions` (shown as method icon)	
`SAP table class` (shown as SAP table icon)	
`SAP structure class` (shown as class icon)	

Solution Customizing

These options may be set on the `SAPProxy sapwsdl` file in the designer. Based on this input the SAP proxy C# classes will be updated automatically.

`Beautify`	If turned on, all identifiers that do not have custom names are “beautified” to mixed case
`ClassName`	The name of the class that is generated
`CreateAsyncs`	Controls if the generator should create Microsoft style asynchronous methods
`CreateTRFC`	Controls if the generator should create methods for Transactional RFC
`CreateQRFC`	Controls if generator should create methods for Queued RFC
`ProxyType`	Whether the generator should create a client proxy or server stub

Method Customizing

These options may be changed on the RFC function (proxy methods):

`Name`	The name used in the code to identify the RFC
`Custom Parameter order`	In the parameters collection editor you can customize the parameter order. If you do so, this property is set to true. If you set it to false, the parameter order is set back to default
`Exceptions`	Use the collection editor to see and customize the ABAP exceptions that this method might throw
`Parameters`	Use the collection editor to see and customize the parameters of the method

UseIOStructs	When turned on, the generator will create an additional version of the method that takes an input structure and returns an output structure (similar to the SAP Java Connector). You cannot have removed parameters with this option

These parameters of the method are provided as informational:

AbapName	Contains the original name of the object as it is called in ABAP
OptionalParameters	Shows you the number of optional parameters in the function
RemovedParameters	Shows you the number of parameters that have been removed and will not be generated
TotalParameters	Shows you the total number of parameters of the function

SAPStructure Customizing

These parameters may be changed on the SAP Structures:

Name	The name of the SAP structure
Fields	Uses the collection editor to customize the fields in the structure
Filename	Specifies the filename that is used to create the type into

These parameters are provided as informational:

AbapName	Contains the original name of the object as it is called in ABAP

SAPTable Customizing

These parameters may be changed on SAPTables:

BaseStruct	For implicitly defined tables, the property contains the name of the structure that the table is based on
Filename	Specifies the filename that is used to create the type into

These parameters are provided as informational:

AbapName	Contains the original name of the object as it is called in ABAP

2.4 SAPClient Methods

Public Constructors

`public SAPClient (SAP.Connector.Destination destination)`	Initializes a new instance of the `SAPClient` class with the connection information from a SAP Destination object
`public SAPClient (System.String ConnectionString)`	Initializes a new instance of the `SAPClient` class with the connection information from a connection string. The connection string can be created manually or from one of the SAP logon controls. If you want to create a connection string manually, see the SAP Remote Function Call API documentation (`RFCOpenEX`).
`public SAPClient ( )`	Initializes a new instance of the `SAPClient` class without connection information. The connection information has to be set in a separate step before you can use the proxy

Public Methods

`Public void <RFC Name> (RFC parameters)`	Executes a normal RFC call for the function specified in the SAP system. Typically out parameters are passed by reference
`Public void TRFC<RFC Name> (RFC parameters, RFCTID)`	Executes the RFC specified as TRFC and therefore requires an `RFCTID` parameter. You must have selected *Create TRFC* for this method to be available
`Public void QRFC<RFC Name> (RFC parameters, RFCQueueItem)`	Executes the RFC specified as QRFC and therefore requires a `RFCQueueItem` parameter. You must have selected *Create QRFC* for this method to be available
`Public System.IAsynchResult Begin<RFC Name> (RFC parameters, System.AsynchCallback, object asynchState)`	Executes the RFC specified using a Microsoft style asynchronous method invocation
`Public void End<RFC Name>(system.iasycresult asyncresult, ref parameters)`	Used to reattach to the result from the Asynchronous method invocation
`public System.Boolean CommitWork ( )`	Used for stateful BAPI calls to commit the logical unit of work to the system
`public void ConfirmTID (SAP.Connector.RfcTID tid)`	When using TRFC or QRFC programming you can use this method to confirm the TID after successfully passing the TRFC call to the SAP system
`public System.Boolean RollbackWork ( )`	Used for stateful BAPI calls to commit the logical unit of work to the system

Public Properties

`public SAP.Connector.SAPConnection Connection [ get, set ]`	The connection object for connecting to the SAP system. Can be either SOAP or RFC connection. Should be created from one of the SAP logon controls
`public System.Int32 RfcTotalMiliSeconds`	The execution time in milliseconds

Private Methods

`protected System.IAsyncResult BeginSAPInvoke (System.String method , object[ ] methodParamsIn , System.AsyncCallback callback , System.Object asyncState)`	Used internally to start the asynchronous method invocation
`protected object[ ] EndSAPInvoke (System.IAsyncResult ar)`	Used internally to receive the results of the asynchronous method invocation
`protected object[ ] SAPInvoke (System.String method , object[ ] methodParamsIn)`	Used to submit a normal RFC for execution
`public void tRfcInvoke (System.String method , object[ ] methodParamsIn , SAP.Connector.RfcTID tid)`	Used to submit a TRFC for execution
`public void qRfcInvoke (System.String method , object[ ] methodParamsIn , SAP.Connector.RfcQueueItem qItem)`	Used to submit a QRFC for execution
`public void ActivateQueue (System.String QueueName)`	Used with QRFC processing
`public void DeativateQueue (System.String QueueName)`	Used with QRFC processing

2.5 SAPClient Exceptions

Errors in the .NET Connector are thrown as .NET exceptions. An exception is the preferred method to handle errors because exceptions are harder to ignore than are return codes. They also provide you detailed information to create more robust applications.

The SAP .NET Connector has the following exception classes:

- RfcCommunicationException
- RfcLogonException
- RfcSystemException
- RfcAbapException
- RfcException
- RfcMarshalException

Coding Recommendations for RFC Exceptions

We recommend you to have at least two `catch` blocks. The first catch statement is for SAP exceptions (specific) and the second for others (generic), for example errors from the runtime or other resources.

We recommend you to close the proxy connection in the `finally` programming block.

SAP .NET Connector closes connections eventually. However, to achieve better performance, we recommend you to close the connection in the `finally` clause of your class. We also recommend that you close any external resources such as open files or database connections here as well.

Instead of providing a status of the SAP RFC connection, we recommend that you simply invoke the method and deal with the exception. The SAP .NET Connector will maintain the status of the connection internally.

The `Proxy.Connection.Open()` method causes an RFC ping. This allows you to see if the system is up. Subsequent `Proxy.Connection.open()` methods will be ignored until there is a `Connection.Close()`.

Example

```
try
      {
            // Call methods here
            proxy.Connection.Open();
            proxy.Rfc_Function_Search(txtRFC.Text,"EN", ref tblRFC);
      }
catch (RfcCommunicationException ex)
      {
            MessageBox.Show(ex.ToString());
            return;
      }
catch (RfcLogonException ex)
      {
            MessageBox.Show(ex.ToString());
            return;
      }
catch (RfcAbapException ex)
      {
      switch (ex.AbapException)
        {
      case (SAPProxy1.No_Function_Found):
            MessageBox.Show("no function found");
            break;
      case(SAPProxy1.Nothing_Specified):
            MessageBox.Show("Nothing specified");
            break;
      default:
            MessageBox.Show("Some unknown abap error occurred");
            break;
        } //switch
      }
catch (Exception ex)
      {
            MessageBox.Show(ex.ToString());
            return;
      }
finally {proxy.Connection.Close();}
```

2.5.1 RfcCommunicationException Class

```
[C#]
public class RfcCommunicationException : SAP.Connector.RfcException
```

This exception class represents an application exception when an RFC communication error occurs. This happens, for example when the .NET Connector cannot connect to an SAP system. Possible reasons are:

- Incorrect server or instance specified in the logon string
- Server is not accessible:
  - SAP Service is not started on that server
  - Client is offline
  - Network problems

This exception is typically thrown at the `proxy.Connection.Open()`; method. It may also occur on any subsequent method invocation, for example an RFC invoke, if the connection is broken.

When this exception occurs, the RFC handle is not created or is no longer valid by the connector. Therefore there is no need to call `proxy.Connection.Close()`;

The following example shows an `RfcCommunicationException`:

```
SAP.NET.Connector.RfcCommunicationException: Connect to SAP gateway
failed
Connect_PM  GWHOST=Iwdf901, GWSERV=sapgw00, ASHOST=Iwdf901, SYSNR=00
LOCATION    CPIC (TCP/IP) on local host
ERROR       hostname 'Iwdf901' unknown
TIME        Mon Apr 29 15:11:51 2002
RELEASE     620
COMPONENT   NI (network interface)
VERSION     36
RC          -2
MODULE      ninti.c
LINE        382
DETAIL      NiPHostToAddr
SYSTEM CALL gethostbyname
COUNTER     15
```

`RFCCommunicationException` derives from `RFCException` and therefore implements the same properties.

2.5.2 RfcException Class

`RfcException` is the base class for the other `SAP RFC Exception` classes. This exception will not be raised by the connector under normal conditions.

```
Object, ISerializable
    System.Exception
       System.ApplicationException
           SAP.Connector.RfcException
```

```
[C#]
public class RfcException : System.ApplicationException
```

Properties of this class

`ErrorCode`	The error code from the SAP application. Derived from `RFCException`
`ErrorGroup`	The error group from the SAP application. Derived from `RFCException`

2.5.3 RfcAbapException Class

This class represents an exception raised by the ABAP program in the SAP system. Exceptions are a type of parameter for each SAP RFC. The connector provides strongly typed support for each RFC's ABAP exceptions in the proxy.

```
[C#]
public class RfcAbapException : SAP.Connector.RfcException
```

In addition to the properties provided from `RfcException`. `RfcAbapException` provides the following property:

`AbapException`	A string containing the ABAP exception from the RFC

Example

In the SAP system you can navigate to the Function Builder (transaction code `se37`) and examine the exceptions for the function you wish to call. For `RFC_FUNCTION_SEARCH` you see the following exceptions:

- `NOTHING_SPECIFIED` – this occurs when no input is specified.
- `NO_FUNCTION_FOUND` – this occurs when no function matches the search selection.

Depending on what the ABAP exception is we might do different things. Therefore a switch statement is a good idea to deal with this exception.

```
catch (RfcAbapException ex)
      {
      switch (ex.AbapException)
        {
      case (SAPProxy1.No_Function_Found):
            MessageBox.Show("no function found");
            break;
      case(SAPProxy1.Nothing_Specified):
            MessageBox.Show("Nothing specified");
            break;
      default:
            MessageBox.Show("Some unknown abap error occurred");
            break;
        } //switch
}
```

2.5.4 RfcLogonException Class

This exception is thrown when the SAP service is available but the user name or password is not accepted by the SAP system.

```
[C#
public class RfcLogonException : SAP.Connector.RfcException
```

Remarks

The most typical causes for this are:

- Incorrect user name or password specified by the user
- License has expired on the SAP system
- The user's account is locked or expired.

Example

```
SAP.NET.Connector.RfcLogonException:

Name or password is incorrect. Please re-enter
```

2.5.5 RfcSystemException Class

This exception is not as typical as the others but may occur from time to time. This exception class is thrown when a short dump related to the current context has occurred in the SAP system. This might be due to a program error in the SAP system or some other exceptional error that occurred on the SAP system.

```
[C#]
public class RfcSystemException : SAP.Connector.RfcException
```

2.6 Debugging of SAPClient Proxies

With the SAP .NET Connector you can debug from your C# proxy directly into the SAP function module by setting the ABAP_DEBUG flag in the connection string. The best way to do this is to set the AbapDebug and Trace flags on the SAPLogonDestination object. Debugging through to the SAP system is useful when you are getting unexpected results back from the SAP system. To use the ABAP_DEBUG option you must have installed SAPGUI on your developer workstation. You cannot use the integrated ABAP debug option with web applications because they run under another Windows context that is invisible to the interactive user. For that reason it is often useful to test your proxy first against a Windows form or Console application in case integrated debugging is required.

It is often useful to run the SAP RFC function directly in the SAP system using transaction SE37. It is easier to isolate the problem once you are sure that input values are valid. In addition, you can use the SAP function module documentation. The SAP data dictionary, which is integrated in the function editor, also gives you information on valid input values.

In the SAP system, there are automatic formatting functions that are not available in the connector, for example to add leading zeroes to a customer or invoice number. If the function is working in the SAP system but not in your proxy, it could be that the SAP system has applied one of these automatic formatting routines but you did not.

To see what values the SAP function module is using, proceed as follows:

1. Execute transaction SE37.
2. Enter the function name and select *Single test* (F8).
3. Enter valid parameters and select *Debugging* (Ctrl F7).
4. Use *Single Step* (F5) to examine the code execution and variable values in the SAP function debugger window, part of the Abap/4 developer workbench.

You can turn on tracing in the SAP system by setting the environment variable CPIC_TRACE and RFC_TRACE. For more information on RFC tracing, refer to **SAP Note 65325**.

CPIC_TRACE writes out detailed trace files to the application directory so it is not advisable to leave tracing on any longer than you need it.

You can do advanced tracing within the SAP system, for example from transaction code SM50 and by examining the work process trace files (for example, dev_w0 in the SAP work directory).

We recommend developers to use the SAP .NET Connector exception classes in their code to determine why an error has occurred and to use as a starting point for debugging.

TRFC applications can be monitored from the TRFC monitor (transaction SM58). QRFC applications can be monitored from QRFC monitor (transaction SMQ2).

2.7 Authentication

The SAP Connector can support all SAP authentication mechanisms including:

- User Name and Password
- X.509 Certificates
- External authentication (for example Microsoft Passport)
- Secure Network Communications (for example, Kerberos and NTLM)

And in addition the SAP login form uses standard ASP .NET forms authentication to provide an alternative single sign-on capability.

The following logon controls are available to manage the SAP authentication in your project.

`Destination`	The base class that holds connection information as properties but does not implement logic to retrieve the connection information
`SAPLogonDestination`	A `Destination` component that retrieves information from the SAPGUI (`SAPLOGON.INI`). Derived from `Destination`
`SAPLoginProvider`	Used with ASP .NET applications to provide forms based logon and an alternative single sign-on mechanism with session state and ASP .NET cookies
`SAP Login Form`	Used with ASP .NET applications to provide forms based logon and an alternative single sign-on mechanism

2.7.1 User Name and Password

The SAP system can make use of various single sign-on options. However, many customers still use a separate user name and password for logging on to SAP. When designing SAP .NET Connector applications, we recommend to avoid scenarios where a single user ID is used to connect to the SAP system. In general, all users of the SAP system whether through SAPGUI or the SAP. NET Connector must be licensed. For that reason, it is preferable to ask the user for their SAP user name and password as part of your application or to make use of one of the single sign-on options.

A very good option for user name and password authentication is the SAP Login form. If you are writing an ASP .NET application this provides you with a forms-based authentication and the necessary plumbing for both session and cookie based single sign-on after the first time visiting the form.

2.7.2 Single Sign-On

Single Sign-On (SSO) is a simplified method of logging on to the SAP system without reducing security. When a system has been configured for Single Sign-On, an authorized user who has logged on to the operating system can access the SAP system simply by selecting it in the SAP logon window or clicking on the shortcut. The user is authenticated by Windows or some other trusted authority so no SAP password is required. All SAP supported single sign-on options are also supported by the connector.

Single Sign-On Technology	Scenario to Use
`Kerberos`	When client has SNC Kerberos library. Use with rich client applications to provide single sign in same way as SAPGUI. Can also be used with ASP .NET impersonation
`NTLM`	When client has SNC NTLM library. Use with rich client applications to provide single sign in same way as SAPGUI. Can not be used with ASP .NET impersonation
`X.509 Certificates`	Web scenarios – especially internet scenarios where client is not inside the firewall
`Microsoft Passport`	Same as X.509 certificates
`SAPSSO2`	With Enterprise Portal
`SAPLoginForm`	User name and password authentication in an ASP .NET web form can be used for all web scenarios
`SAPSSO1`	With older ITS scenarios

2.7.2.1 X.509 Certificates

In this scenario you must first configure a working SSO account for the logged-on user (for example ASPNET or other Windows user that your application runs under). The SAP system examines the X509 certificate to determine an external user ID. This external user is logged on to the SAP system. This allows you to only have to setup an SSO connection between the web server and the SAP system and then using the certificate field to map from the X509 certificate to the SAP user.

Alternatively, you can use Active Directory or IIS mapping and impersonate the user. This method is discussed in part in the section on Kerberos or NTLM SSO with impersonation above.

In the connector, the `SAPLogonDestination X509Certificate` property should contain the value of the X509 certificate BASE64 encoded. This should be set at runtime after reading the contents of the X509 certificate from the user's browser.

Certificate Field	Returns
`Request.ClientCertificate.Subject`	The subject that is mapped to SAP external user ID in table `VUSREXTID`. For example: `CN=SAPDotNet`
`Request.ClientCertificate.Certificate`	A byte array containing the binary stream of the entire certificate content. You must use the `Convert.ToBase64String` function to format it for sending to SAP system

The following example shows code for using the browser certificate in `SapLogonDestination`:

```
sapLogonDestination1.X509Certificate =
Convert.ToBase64String(Request.ClientCertificate, 0,
Request.ClientCertificate.Length);
```

Do not use USER parameter in your Destination component with X.509certificate logon.

Setting Up Certificate Mapping to SAP User in the SAP System

The procedure consists of the following steps:

- You enable an SNC connection between IIS and the SAP system with transaction `SNC0`.
- You map the certificate to SAP table `VSUSREXTID`.

Input for Setting Up Certificate

Step	Option	Explanation
Enable an SNC connection between IIS and the SAP system. Use transaction SNC0 to update the *Access Control List* (ACL)	System ID	Enter your SAP system ID.
	SNC name	Enter the Secure Network Communications (SNC) user name. For example for Kerberos enter: p:<SAPService_User@<DOMAIN_NAME> For more information on SNC names refer to the *SNC User Guide* at the SAP Service Marketplace at **http://service.sap.com**
	Entry for RFC ... Entry for ext. ID	Activate all entries for RFC, CPIC, DIAG, certificate, ext. ID
Map the certificate to the SAP user with transaction SM30	External ID type	a. Maintain table VUSREXTID b. Enter DN for External ID type

Step	Option	Explanation
	External ID	Enter the ID exactly as stated in the certificate, for example CN=SAPDotNet To find out the Subject name (External ID) you have the following options: • **Using the Internet Explorer** a. In the Internet Explorer, choose *Tools* → Internet Options → *Content* → *Certificates.* b. Select the certificate and choose *View.* c. On the Certificate screen, choose *Details*. d. Go to *Subject* to see the name. • **Using SAP Process Tracing** a. Use transaction SM50. b. Turn on tracing for the component SECURITY and use trace level 2: – Choose *Process* → *Trace* → *Display settings* → *Display Components* and select SECURITY. – Choose *Process* → *Trace* → *Dispatcher* → *Change Trace Level* and enter 2 for the trace level. Now tracing is enabled on the SAP application server. c. Run the .NET Connector application that is using the x.509 certificate to connect to the SAP system. d. On the SAP application server search for a file named dev_wp<workprocess_number> (for example, dev_w0) that contains the text string CertGetInfo. The subject name is next to the text string.
	User	Enter your SAP user name.

Setting Up Certificates in IIS:

For certificates to work you have to configure IIS to use HTTPS.

Here is an example for an ASPX page code to test whether certificates are working in IIS:

```
User (from Context):  <%=Context.User.Identity.Name%> <P>
User (from Thread):
<%=System.Threading.Thread.CurrentPrincipal.Identity.Name%><P>
Certifcate: <%=Request.ClientCertificate.Subject%>
```

For more information about using X.509 certificates in Windows 2000, refer to the *Step-by-Step Guide to Mapping Certificates to User Accounts* at http://www.microsoft.com/windows2000/techninfo/planning/security/mappingcerts.asp

2.7.2.2 Microsoft .NET Passport

You can use external authentication mechanisms such as Microsoft .NET Passport to determine which user to log on to the SAP system. This mechanism is similar to using X.509 certificates as discussed above. Instead of asking the SAP system to examine the certificate, we determine the user identity through some other means (for example the Passport API). Over a trusted SNC connection to the SAP system we tell the SAP system which user to log on. It is important that the IIS service user is a trusted SNC user in the SAP system. For more information, see to the section on establishing an SNC connection between IIS and the SAP system above.

The `EXTIDDATA` tag is the external user ID as defined in view `VUSREXTID` for type `ID`.

The tag `EXTIDTYPE` should be equal to `ID` as that is the generic external user type. Other external user types, for example type `DN` (certificates) do not work in this scenario.

```
// Example connection string for passport type authentication
string ConnStr = "ashost=pcintel11 sysnr=0 client=0 snc_mode=1
snc_partnername=\"p:SAPServiceCS2@nt5.sap-ag.de\" type=3
EXTIDDATA=<passport_id> EXTIDTYPE=ID";
```

To verify that the external user really has logged on, set `abap_debug = 1` in the connection string. Then examine the list of logged on users in the *SAP users overview* screen, which you can access in the SAP menu under *System Monitoring*. Alternatively, use transaction `SM04`.

In the future, both SAP and Microsoft will offer more direct support for Passport authentication. For more information, refer to *Microsoft's Federated Security and Identity Roadmap* at: `http://msdn.microsoft.com`

2.7.2.3 Kerberos and NTLM

In the Windows environment it is possible to use Kerberos, NTLM and X509 certificates as single sign-on options. The client where the proxy is running must have the appropriate GSS library (for example `gsskrb5.dll`) and the correct environment variables set. For more information, see the *SNC Users guide* and the *SAP Web AS Inst. on Windows: MS SQL Server* at the SAP Service Marketplace at: **`http://service.sap.com`**.

Before configuring the SAP .NET Connector proxies to use SSO we recommend you to test the connection with SAPGUI to be sure single sign-on is working. For technical reasons you must still provide the SAP user name in the connection string. In many companies mapping of the SAP user to the Windows NT user is quite easy, as for example they use the same name or apply a logical naming process. The exception to this is with X509 certificates where no USER parameter should be provided.

To connect to the SAP system with the SNC parameters, you can use one of the SAP Destination components. You can also construct your own connection string; however, due to deployment problems we do not recommend hard coding correction strings.

Here is an example of a connection string using Kerberos to show what parameters to set on your destination component.

```
// connection string with SNC parameters and debug
string ConnStr = "ashost=pcintel11 client=000 snc_mode=1 sysnr=00
type=3 user=SAPDOTNET snc_partnername=\"p:SAPServiceCS2@nt5.sap-
ag.de\";
```

The disadvantage of SSO is that you may have to configure additionally each client machine. However, you can use Active Directory to distribute the SSO configuration to users. For more information, see the installation guide *SAP Web AS Inst. on Windows:MS SQL Server* on the SAP Service Marketplace at: **`http://service.sap.com`**.

An alternative to configuring all user machines for SSO is to use a web application, impersonate the user, and then perform SSO as that user. In this case, you only have to configure the web server. You also have to configure your IIS application for impersonation. For more information about ASP .NET impersonation, refer to the *.NET Framework Developer's Guide: ASP .NET Impersonation* at:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconaspnetimpersonation.asp

When Using ASP .NET Impersonation

Be sure to set these values in the authentication section of the `Web.config` file:

```
<identity impersonate=”true” />
<authentication mode = “Windows” />
```

In *IIS Administration* turn off *Anonymous* access under the directory security tab

A simple ASPX page to test if impersonation is working is:

```
User Name: <%=System.Environment.UserName />
Domain: <%=System.Environment.UserDomainName/>
```

2.7 Authentication

When you access this test web page and if impersonation is working, you can see the logged-on NT user but not the IIS anonymous user.

There is currently technical limitation in the Kerberos implementation from SAP. You can only use Kerberos with the client machine at this time.

Kerberos is case sensitive so make sure that you get SNC_PARTNERNAME correct. The SAP user name is not case sensitive. This is only needed due to a limitation in the RFC library and not because of SSO considerations.

NTLM is not supported in this web scenario because it does not provide impersonation capabilities.

2.7.2.4 Destination Class

This is the base SAP Destination component that can be used to set connection and authentication properties. This class does not contain any logic to retrieve the login properties but other classes that derive from it do (for example `SAPLogonDestination`).

```
[C#]
public class Destination : System.ComponentModel.Component
```

Public Properties

`AbapDebug`	Set to true to use integrated ABAP debugging
`AppServerHost`	The name of the SAP application server
`Client`	The number of the SAP client you are connecting to
`ConnectionString`	Read only property containing the connection string data used to connect to the SAP system
`ExtIdentificationData`	Used with external authentication scenarios (for example, Microsoft Passport)
`ExtIdentificationType`	Used with external authentication scenarios (for example, Microsoft Passport)
`Language`	Optional
`LogonGroup`	If LogonGroups are used enter it here
`MsgServerHost`	If a message server is required to connect to the SAP system enter it here
`MySAP_SSO`	The older style SAP single sign-on cookie used by the Internet Transaction Server.
`MySAP_SSO2`	The newer SAP Logon ticket used by the Enterprise Portal
`Password`	If not using SSO enter the password here
`SAPSystemName`	Read only property showing the three digit system SID
`SNCLib`	If using Kerberos or other SNC library enter the path here or alternatively use an environment variable as described in the *SNC User Guide* on the SAP Service Marketplace
`SNCMode`	If set to true SNC is used
`SNCMyName`	Not required
`SNCPartnerName`	The SNC name of the application server service user (for example, `p:SAPServiceCS2@nt5.sap-ag.de`)
`SNCQoP`	Optional. See *SNC User Guide* for more information about QOP with SNC
`SystemNumber`	The SAP system number (for example 00)

`Trace`	If true, detailed trace files are written to the application directory
`Type`	A string showing the data type of the destination object
`UserName`	The SAP user name
`X509Certificate`	The base64 encoded contents of the x509 certificate

Remarks

The destination object is used within the SAP Login Form for forms-based authentication.

2.7.2.5 SAPLogonDestination Class

The `SAPLogonDestination` class is a design time component derived from the Destination class. Connection information can be retrieved from the SAPGUI at runtime, specifically from SAPLOGON.INI on the machine hosting the application.

```
[C#]
public class SAPLogonDestination : SAP.Connector.Destination
```

Public Properties

`public string DestinationName [ get, set ]`	Provides a drop down list of the destinations stored in the SAPGUI. This is the same list the user would see when starting SAP Logon for example

Private Properties (Retrieved from SAPGUI)

`public string AppServerHost [ get, set ]`	The application server host
`public System.Collections.IDictionary AvailableDestinations [ get]`	The list of available destinations in a name/value pair
`public string DestinationDescription [ get, set ]`	A description of each destination
`public string LogonGroup [ get, set ]`	Logon group if configured
`public string MsgServerHost [ get, set ]`	Message server host if configured
`public string SAPSystemName [ get, set ]`	SAP System name (for example CS2)
`public bool SNCMode [ get, set ]`	Whether SNC is used
`public string SNCPartnerName [ get, set ]`	The SNC name of the SAP application server as stored in SAPGUI. See *SNC User Guide* for details
`public short SystemNumber [ get, set ]`	The system number. For example 00.
`public string Type [ get, set ]`	The data type.

Public Methods

`public System.String GetDestinationNameFromPrintName (System.String printName)`	Retrieves the destination name from the print name

Remarks

This component is designed to simplify the SAP Logon process for both client applications where the user already has a SAPGUI and for web applications where administrators wish to store connection information in the SAPGUI instead of in the destination object or dynamic property in the `webconfig.xml` file.

Use of SAPLogonDestination with SAP Router

If your SAPGUI destination has a router, carry out the following steps to use this component:

1. Create a new destination in SAP Logon.
2. On the Application server field put the SAP router string first, then the application server.
3. Leave the SAP Router String blank.
4. Test it in SAPLogon before using with the connector.

For example, if you have the following:

```
Description: MySystem
Application Server: IWDF9387.WDF.SAP.CORP
SAP Router String: /H/SAPGATEA.WDF.SAP-AG.DE/S/3291/H/
```

In your new destination, the properties should be:

```
Description: MySystem_DNC
Application Server:
/H/SAPGATEA.WDF.SAP-AG.DE/S/3291/H/IWDF9387.WDF.SAP.CORP
SAP Router String:
```

2.7.2.6 SAPLoginProvider Class

The `SAPLogonProvider` class can be used with ASP .NET applications to retrieve a connection to the SAP system from the SAPLogin form or if available from either the session state or an ASP .NET cookie if the user had already visited the site. See SAP Login Form [Page 52] for details. The `SAPLoginProvider` is a static object of the SAP .Connector.

```
[C#]
public class SAPLogonProvider : System.Object
```

Public Properties

`Connection`	Not yet implemented

Public Methods

`public static void CloseSAPConnection (System.Web.UI.Page` *p*`)`	Used internally to close the SAP Connection
`public static SAP.Connector.SAPConnection GetSAPConnection (System.Web.UI.Page p)`	This method is used to get the connection from a `SAPLoginForm` in your project. It is part of the SAP Connection code provided in the SAP Proxy toolbox
`public static void OpenSAPConnection (System.Web.UI.Page p , System.String connstr , System.Boolean persist)`	Used internally to open the SAP Connection

Example

```
proxy.Connection =
SAP.Connector.SAPLoginProvider.GetSAPConnection(this);
```

2.7.2.7 SAP Login Form

This ASP .NET web form can be added to your project with *Project* → *Add New Item*. Look for the Web Project Items and select *SAP Login Form.*

For technical reasons it is better to leave the name of this form as `SAPLogin1.aspx`. The design of this form can be changed in the designer to suit your requirements. The purpose of this form is to provide SAP user name and password authentication for ASP .NET applications built with the connector.

```
[C#]
Public class SAPLogin1.aspx : System.Web.UI.Page
```

Remarks

When you select *Save*, a cookie is written to the hard disk with a name like user@server.txt

The login information is stored in an encrypted format in the ASP .NET cookie. When the user logs on the next time, the ASP .NET application will look in this cookie to retrieve the logon information for the SAP system.

The SAPLogin form has a built-in destination component to store connection information and when the login information is entered it will be stored here as well. Default connection settings are stored here based on what values were used in the proxy generation wizard.

The destination component uses the web config file to store connection information as dynamic properties. A blue icon next to the property in the Destination control object shows that the property is synchronized with the web config file.

We recommend that the main page of your application be called `default.aspx` as this is the default redirect for the SAP Login page. For technical reasons, the SAP Login page should always be called `SAPLogin1.aspx`. It is possible to change this but you must then also update the `web.config` file *Authentication mode* section which by default points to a `loginUrl` of `SAPLogin1.aspx`.

After successfully logging in for the first time on this page, the login form does a redirect back to the original page (for example, `default.aspx`). If the login page is called directly instead of as a redirect

If reached without original form then it navigates to `default.aspx`. Therefore, you should have a `default.aspx` page in your application.

When the `SAPLogin1.aspx` form is called, it executes the following logic:

First, it looks in the session state and cookies to find out if this session connection information is already known. If so, it tries to open SAP connection using the static `openConnection` method. If this is not successful it tells the ASP .NET provider it´s not working by raising an exception.

After successful login it stores the active connection object in session state with special name. What is stored in the session state is the actual connection.

If you selected the *Save* option, a cookie is stored to your hard disk that has encrypted complete connection string.

In this way, the SAP Login form provides an alternative connection pooling and single sign-on where many distinct SAP accounts are hitting the site.

When multiple sessions use the same user ID, the connector provides a dedicated connection pool object.

2.8 SAPIDocSender Class

The SAPIDocSender class is a TRFC client used to submit SAP intermediate documents to an SAP system for later processing. An example of an IDOC might be a customer sales order or a material master record. IDOC records are typically used in EDI scenarios. See the IdocSubmit sample for example code.

```
[C#]
public class SAPIDocSender : SAP.Connector.SAPClient
```

Public Constructors

`public SAPIDocSender (System.String ConnectionString)`	Creates a new instance of SAPIDocSender with a connection string
`public SAPIDocSender ( )`	Creates a new instance of SAPIDocSender and set the connection in a later step

Public Methods

`public void SubmitIDoc (System.String iDocPath , SAP.Connector.RfcTID tid)`	Submits an IDOC from an IDOC stored as a file on the operating system
`public void SubmitIDoc (System.IO.TextReader iDoc , SAP.Connector.RfcTID tid)`	Submits an IDOC from an IDOC as a textreader object. Perhaps from another application or built dynamically
`public void TRfcIDocInBoundAsynchronous (SAP.Connector.EDI_DC40_BLOCKList iDocControlRec40 , SAP.Connector.EDI_DD40_BLOCKList iDocDataRec40 , SAP.Connector.RfcTID tid)`	Offers more granular control over the different pieces of the IDOC (EDIDC and EDIDD) for example when you are creating an IDOC manually or changing something in the header but otherwise want to keep the body of the IDOC

Remarks

An IDOC will consist of three segments, the header (EDIDC), the body (EDIDD) and the status (EDIDS). The EDIDC record can contain two formats (EDIDC or EDIDC40) depending on the version of the IDOC. The body of the IDOC will differ depending on what IDOC type and release it is. The status record is only maintained inside of the SAP system and is not relevant to submit an IDOC.

Example

```
private SAP.Connector.SAPIDocSender sapiDocSender1;
private void SubmitIdoc()
{
// submit to SAP via trfc
RfcTID myTid = RfcTID.NewTID();
try
{
sapiDocSender1.ConnectionString = destination1.ConnectionString;
```

```
sapiDocSender1.SubmitIDoc(@”C:\temp\idoc.txt”,myTid);
sapiDocSender1.ConfirmTID(myTid);
MessageBox.Show("Idoc was submitted to SAP, look at transaction WE02 in
SAP","idoc status");
}
catch (Exception ex)
      {
      MessageBox.Show("Problem submitting IDOC to SAP\n" +
ex.ToString());
       }
} //submitIdoc()
```

2.9 Asynchronous Methods

With .NET Connector, we can take advantage of many features of the CLR including support for easy asynchronous programming. This powerful feature is not available on any other SAP connector at this time. In all other connectors, BAPI and RFC calls are synchronous calls.

When we use asynchronous methods from the .NET Connector wizard, the proxy contains two additional methods `Begin<RFC name>` and `End<RFC Name>`.

When `Begin<RFC name>` is called, CLR queues the request and returns immediately to the caller. The target method will be on thread from the thread pool. The original thread is free to continue executing in parallel to the target method. If a callback has been specified on the `Begin<RFC name>`, it will be called when the target method returns. In the callback, the `End<RFC name>` method is used to obtain the return value and the `in/out` parameters. If the callback was not specified on the `Begin<RFC name>`, then `End<RFC name>` can be used on the original thread that submitted a request.

When working with asynchronous calls, we need three additional variables compared to a synchronous RFC call.

Variable	What It Does
`System.IAsyncResult asyncresult`	A return of `IAsyncResult` is required to implement Asynchronous Method signatures. The result of the call (`IAsyncResult`) is returned from the begin operations, and can be used to obtain status on whether the asynchronous begin operations has completed. The result object is passed to the end operation, which returns the final return value of the call
`System.AsyncCallback callback`	A delegate class that is called when the operation has completed. If null, no delegate is called. We can either use a callback delegate as shown below or pass the `AsyncCallback` delegate as null. In that case, no delegate will be called and we have to check the status ourselves. To check the status we can check the `AsyncResult.IsCompleted` property or if we are using a callback, this delegate will be called automatically when the method has completed
`object asyncState`	Extra information supplied by the caller

Example

The following Winform sample shows the code for asynchronous method call (SAPAsyncSearch method).

```
private void SAPAsyncSearch()
/* this routine calls RFC_CUSTOMER_GET using .NET asynchronous
 * method invocation. When the function is completed asynchronously
in SAP,
 * the function "myfunction" is called. */
SAPConnect();
myAsyncState = null;
myCallback = new System.AsyncCallback(myFunction);
asyncresult = null;
try
{
asyncresult = proxy.BeginRfc_Customer_Get(g_custNo, g_custName, ref
brfcknA1Table1, myCallback, myAsyncState);
}
catch (Exception ex)
{
MessageBox.Show("Error returned in Async search\n" + ex.ToString(),
"SAP Async Search problem");
return;
}
}
```

See also:

.NET Framework Developer's Guide: Asynchronous Programming Overview

.NET Framework Developer's Guide: Including Asynchronous Calls

.NET Framework Developer's Guide: Asynchronous Delegates

.NET Framework Developer's Guide: Asynchronous Method Signatures

.NET Framework Class Library: SoapHttpClientProtocol Class

2.10 TRFC Client Programming

When calling an RFC as a transactional RFC (TRFC) there are no return values. If the submission for some reason does not work, an exception will be raised. The TRFC method requires an additional `RfcTID` parameter. You should let the SAP system know to confirm this TID if the submission is successful. TRFC submissions are used when you require guaranteed one time only execution of a function but do not require any return information beyond that the call was accepted by SAP. Therefore TRFC is best for submitting data but not for retrieving data.

Example

The Winform `SAPUpdateTRFC` method gives an example of TRFC coding.

```
private bool SAPUpdateTRFC()
      {
/* this routine updates the customer list to SAP using (T)RFC */
RfcTID myTid = SAP.Connector.RfcTID.NewTID();
Debug.WriteLine("RFC tid is " + myTid.ToString());
      try
      {
            proxy.TRfcRfc_Customer_Update(ref brfcknA1Table1, myTid);
      }
      catch (Exception ex)
      {
            Debug.WriteLine(ex.ToString());
            return false;
      }
// if you don't confirm the TID, the update is rolled back in SAP.
proxy.ConfirmTID(myTid);
return true;

            } //SAPUpdateTRFC
```

2.10.1 RfcTID Class

The RfcTID class is used with QRFC and TRFC processing. A transaction ID (TID) is a 24 character long unique identifier used in the SAP system. It can be mapped back and forth to a system GUID using the functions described below.

```
[C#]
public class RfcTID : System.Object
```

Public Constructors

`public static SAP.Connector.RfcTID NewTID ( )`	Create a new TID from the SAP .Connector static `NewTID` method
`public RfcTID (System.Guid guid)`	Create a new TID from an existing `System.Guid`

Public Methods

`public System.Guid ToGuid ( )`	Convert from a TID to a GUID
`public virtual System.String ToString ( )`	Convert the TID to a string

2.11 QRFC Client Programming

QRFC is a type of TRFC programming that guarantees the function will be run in a certain order (if and when it is selected to run by the system administrator).

Example

```
private bool SAPUpdateQRFC()
{
/* this routine updates the customer list using (Q)RFC
 * use SMQ2 to see results in SAP */
RfcQueueItem mySAPQueue = new RfcQueueItem("DNCQueue", g_queIndex,
RfcTID.NewTID());
try
      {
      proxy.QRfcRfc_Customer_Update(ref brfcknA1Table1, mySAPQueue);
      }
catch (Exception ex)
      {
      Debug.WriteLine(ex.ToString());
      return false;
      }
      g_queIndex ++; //increment the queue index for next time
      return true;
} //SAPUpdateQRFC
```

2.11.1 RfcQueueItem Class

The `RfcQueueItem` class is used to submit an RFC via QRFC.

```
[C#]
public class RfcQueueItem : System.Object
```

Public Constructors

`public RfcQueueItem (System.String name , System.Int32 index , SAP.Connector.RfcTID tid)`	Creates an instance of an `RfcQueueItem` from a queue name, queue index and RFC tid. If a queue already exists in SAP, the function will be added at that index. If not, a queue will be created

Public Properties

`QueueIndex`	The index to which the function will be added to the SAP queue
`QueueName`	The name of the SAP queue to add to or to create if it does not already exist
`TID`	The TID used to submit the RFC to the SAP system

2.12 Connection Classes

2.12.1 SAPConnection Class

The SAPConnection class is used to manage the connection to the SAP system. Some basis information about the system you are connected to can also be determined from this class.

```
[C#]
public class SAPConnection : System.Object
```

Public Constructors

public SAPConnection (SAP.Connector.Destination dest)	Creates an SAP connection from a destination class
public SAPConnection (System.String connString)	Creates an instance of the SAP connection from a connection string. For example from the SAPlogonDestination or a manually created connection string

Public Methods

Accept	
Close	The Close method should be called after the RFC is completed
Dispose	
Finalize	
Open	The Open method must be called before the RFC can be executed

Public Properties

ApplicationServer	Reads only property showing basis information about the SAP system
CodePageEncoding	Reads only property showing basis information about the SAP system
ConnectionString	Reads only property showing basis information about the SAP system
KernelRelease	Reads only property showing basis information about the SAP system
OwnCodePage	Reads only property showing basis information about the SAP system
PartnerCodePage	Reads only property showing basis information about the SAP system
SystemID	Reads only property showing basis information about the SAP system

`SystemNumber`	Reads only property showing basis information about the SAP system
`SystemRelease`	Reads only property showing basis information about the SAP system

2.12.2 SAPConnectionPool Class

Connection pooling is a more sophisticated way of managing SAP Connections. In a two-tier deployment we do not recommend to use this as each client has its own connection. In an n-tier deployment where multiple users are using the same connection attributes (for example a web site that is accessed by many users) it may make sense to use connection pooling. Alternatively, consider using the `SAPLogin Provider` and `SAPLogin Form`.

```
[C#]
public class SAPConnectionPool : System.Object
```

Public Constructor

`public SAPConnectionPool ( )`	Static object of the SAP .Connector

Public Methods

`public static SAP.Connector.SAPConnection GetConnection (System.String connectionString)`	Gets a connection from the connection pool by passing in the connection string
`public static void ReturnConnection (SAP.Connector.SAPConnection connection)`	Returns the connection to the connection pool

Example

The sample application `DNCWebServiceSample` uses connection pooling.

```
using(proxy.Connection =
SAP.Connector.SAPConnectionPool.GetConnection(this.destination1.Connect
ionString))
```

2.13 Data Binding with SAPTable

SAPTables implement the proper .NET interfaces that allow them to be data-bound to any .NET data aware control such as a data grid, combo-box or list control. To databind to a .NET control set the `datasource` property to the name of the SAP table.

```
dataGrid1.DataSource = BRFCKNA1Table1;
```

In Windows forms there is no need to call `this.Databind()` but you must do so in ASP .NET datagrid to update the datagrid.

Coding Recommendations for SAPTable Parameters in Your Code

- **Pass OUTPUT Tables uninitialized**

  If you do not need all `TABLE` parameters you can remove them from the RFC signature in the Visual Studio .NET designer. This only works when the table is also `OPTIONAL`.

  For example, to call `RFC_CUSTOMER_GET` which has an `OUTPUT` table of customer records, the table variable `tblCust` is passed unitialized.

```
SAPProxy1  proxy = new SAPProxy1();

// create an SAP table but as it's an in parameter don't need to instantiate it for the rfc call
BRFCKNA1Table tblCust = new BRFCKNA1Table();

using(proxy.Connection =
SAP.Connector.SAPConnectionPool.GetConnection(this.destination1.ConnectionString))
        {
        // call the RFC with the signature defined by SAP
        proxy.Rfc_Customer_Get(CustNo, CustName, ref tblCust);
        }
```

- **Fill INPUT TABLES With Values**

  A TABLE is always passed by `REF` and is therefore both an `IN` and `OUT` parameter. Since a table is a collection you can fill it in a loop for example:

```
for(int i = 0; i < number_of_lines; i++)
{
        IDRANGE mySelection = new IDRANGE();
        IDRANGE.Sign = "I"
        IDRANGE.Option = "BT"
        IDRANGE.Low= "0000001000"
        IDRANGETable.Add(mySelection);
}
```

  Alternatively you can create an additional constructor with the parameters you want to fill.

```
for(int i = 0; i < number_of_lines; i++)
```

```
{
        IDRANGETable.Add(new IDRANGE("I","BT","0000001000"));
}
```

For TABLES that are input parameters you have to populate the values yourself before calling the SAP RFC. For TABLES that are output parameters the SAP marshalling code populates the values for you automatically.

- **Clearing Values From the Table**

When an exception is returned, the TABLE object is not reinitialized. If you had made a previously successful method call and the TABLE contains values, these will not be reinitialized after an exception has occurred. In this case you can keep the last (good) results. You can use the table's clear() method to initialize the values yourself.

- **Getting Runtime Information About the RFC TABLES**

SAP Tables are built on the `.NET CollectionBase` class. Therefore they support indexers and other array operations to allow for easy management of the members in the array.

Field metadata information can be determined from the structure using the static method: `GetSAPFieldsSchema(System.Type t)`.

The type parameter is the data type of `SAPStructure` (in this case the type of `BAPICUSTOMER_IDRANGE`).

```
SAPField[ ] myFields =
BAPICUSTOMER_IDRANGE.GetFieldsSchema(typeof(BAPICUSTOMER_IDRANGE));
```

You can get metadata information about each field either with a `foreach` statement or using the `length` property of the `SAPField` array.

2.14 Programming with Visual Basic .NET

SAP Connector proxies are generated in Microsoft Visual C# but you can access them in any common language runtime language (Visual Basic, Smalltalk, Python, Managed C++, etc.). The proxy itself must still be created as a C# project and added as a reference to your project. Within C# projects, you can add the proxy code to the same project. While the samples provided with the connector include the SAP proxy code in each project we do not recommend this approach except to illustrate the concepts of the connector. Normally, you create and compile the proxy class separately from the solutions that use the proxy. For example, it is often a good idea to put both the SAP`.Connector.dll` and the generated proxy into the global assembly cache so that it can be accessed by all projects that require these SAP objects and minimizes the number of proxy components to maintain. Therefore, you can choose which language to use with the SAP .NET Connector.

The SAP toolbox designer components work with both C# and Visual Basic .NET projects. For other languages you may have to add a variable to these classes yourself. The exception to this is the `connect code` template code. This code is designed to work with C# projects so it will not produce syntactically correct Visual Basic .NET code for connecting and calling the proxy functions. To reproduce the template code manually, is very simple. It involves the following steps:

- Declare parameters for the call (for example, a proxy object and a connection)
- Call the RFC method(s)
- Adding error handling (for example, `Try/Catch/finally`)

Example in Visual Basic. NET

```
Private Sub btnSearch_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSearch.Click
    ' Declare parameters here
    Dim proxy As New CompiledProxy.CompiledProxy()
    ' note: compiled proxy is a SAP connector class project in this solution or
    ‘ could be the proxy in the GAC

    ‘ get the connection from SAP destination component (added from toolbox)
    proxy.Connection = New SAP.Connector.SAPConnection(Me.Destination1)

    ' Call methods here
    proxy.Rfc_Customer_Get("", txtSelection.Text, BrfcknA1Table1)

  End Sub
End Class
```

3 SAP RFC Server Programming

To create an RFC server with the SAP .NET Connector, create a new project using *SAP Connector Class*. Select the object type *Server Stub*.

Within the proxy you have to provide the implementation for the function. The SAP system acts as the client, calling your implementation of the function. For example, if you had an RFC server that implemented `BAPI_CUSTOMER_GETLIST` you have to populate the customer address list and special data tables in the `BAPI_CUSTOMER_GETLIST` method of your generated server stub.

The generated server code has methods for calling the function as RFC or TRFC. The methods for TRFC processing are generated for you, for example `CheckTransaction(RfcTID)`, `CommitTransaction(RfcTID)`, `ConfirmTransaction(RfcTID)`, `RollbackTransaction(RfcTID)`.

The SAP Server object has a “host” container to manage one or many RFC Servers. Each RFC server can contain different connection properties (for example, system, gateway, program ID). In the case where the connection information is the same (for example, same program ID), the SAP system calls each RFC Server in a round-robin approach.

This makes it easy to create and manage a multi-threaded, highly responsive RFC server application. When your RFC server application is done, it often makes sense to deploy it as a Windows Service. The `SAPServerHost` object maps very closely to how Windows services are managed, for example, there is a `start`, `stop` and `pause` functionality provided in the `SAPServerHost` or alternatively on each individual server.

3.1 SAPServer Class

The SAPServer class is generated for you by the SAPWSDL file and the SAPConnectorGenerator custom tool in Visual Studio. In addition to the classes which are maintained by the custom tool an additional class SAPProxy1Impl.cs is created as an example and starting point for the server implementation. You can overwrite and modify it as required.

The SAPServer class should be used together with the SAPServerHost class.

```
[C#]
public class SAPServer : System.ComponentModel.Component
```

Public Constructors

public SAPServer (System.String programId , System.String gwhost , System.String sapgwxx , System.String codepage , SAP.Connector.SAPServerHost host)	Creates an instance of SAPServer and adds it to the SAPServerHost instance you specify. You specify the connection parameters individually
public SAPServer (string[ ] args , SAP.Connector.SAPServerHost host)	Creates an instance of SAPServer and adds it to the SAPServerHost instance you specify. You specify the connection parameters in the command line arguments array (args)
public SAPServer (string[ ] args)	Same as above except it does not add the server to the SAPServerHost
public SAPServer (System.String connectionString , SAP.Connector.SAPServerHost host)	Creates an instance of SAPServer and adds it to the SAPServerHost instance you specify. You specify the connection parameters individually
public SAPServer (System.String programId , System.String gwhost , System.String sapgwxx , System.String codepage)	Same as above except it does not add the server to the SAPServerHost
public SAPServer (System.String connectionString)	Creates an instance of SAPServer with a connection string (for example, –a<PROGID> -g<gw host server> -x<gateway service> (optional –c<codepage#>). Does not add the server to a host
public SAPServer ( )	Creates an instance of SAPServer but does nothing else. You have to set connection properties and optionally a host in a separate step

Public Methods

`public virtual System.Int32 CheckTransaction (SAP.Connector.RfcTID tid)`	See SAPServer and TRFC [Page 73]
`public virtual System.Int32 CommitTransaction (SAP.Connector.RfcTID tid)`	See SAPServer and TRFC [Page 73]
`public virtual System.Int32 ConfirmTransaction (SAP.Connector.RfcTID tid)`	See SAPServer and TRFC [Page 73]
`public void Continue ( )`	Resumes the server after being paused.
`public void Pause ( )`	Pauses the server
`public virtual void RollbackTransaction (SAP.Connector.RfcTID tid)`	See SAPServer and TRFC [Page 73]
`public void Start ( )`	Starts the server. This is normally the first step after creating the server(s).
`public void Stop ( )`	Stops the server

Public Properties

`public string ProgramID [ get, set ]`	The program ID as registered on the `SAPGateway` and in the TRFC destination (transaction `SM59`). For example `myProgID`. This is case sensitive!
`public string SAPCodepage [ get, set ]`	Optional - the codepage to use (for example . `4103`). Code page `4103` is Unicode (`UTF16`). If the SAP system you are communicating with is not Unicode (for example release 4.6 Kanji) you may need to change this value
`public string SAPGatewayHost [ get, set ]`	The name of the SAP Gateway host (for example `PCINTEL11`)
`public string SAPGatewayService [ get, set ]`	The name of the SAP Gateway service (for example `sapgw00`)

Example

See the sample `RFCServerConsole` for an RFC server implementation of `RFC_CUSTOMER_GET`.

3.2. Calling our RFC .NET Server from SAP Programs

To execute our .NET server stub application from the SAP system we need to execute the ABAP command **`Call function X Destination Y`**. This report calls our proxy and writes the results to screen. Alternatively, you can use the SAP function module's *single test* capability with the TRFC destination for your .NET server stub.

To create a TRFC destination for the SAP .NET server stub create a destination of type T (TRFC) in transaction code SM59. The program ID in your server stub is case sensitive.

Example

```
*&---------------------------------------------------------------------
*& Report  ZRFCSERVERCALL
*&
**&---------------------------------------------------------------------
*&  This program can be used with the RFCServerConsole sample
*&  Source is available in  %\RFCServerConsole\ABAPProgram
*&---------------------------------------------------------------------

REPORT  ZRFCSERVERCALL                          .
DATA: TBLCUST like BRFCKNA1 occurs 0 with header line.
PARAMETERS: P_CUSTNO like KNA1-KUNNR, P_CUSTNA like KNA1-NAME1,
            P_DEST(15) TYPE C.

CALL FUNCTION 'RFC_CUSTOMER_GET' DESTINATION P_DEST
  EXPORTING
    KUNNR             = P_CUSTNO
    NAME1             = P_CUSTNA
  TABLES
    CUSTOMER_T        = TBLCUST
  EXCEPTIONS
    NOTHING_SPECIFIED = 1
    NO_RECORD_FOUND   = 2
    OTHERS            = 3.

CASE SY-SUBRC.
  WHEN 0.
    LOOP AT TBLCUST.
      WRITE: / SY-TABIX, TBLCUST-KUNNR, TBLCUST-NAME1, TBLCUST-ORT01.
    ENDLOOP.

  WHEN 1.
    WRITE: / 'You need to specify a value ', SY-MSGV1.

  WHEN 2.
    WRITE: / '.NET component didnt find anything ', SY-MSGV1.
  WHEN 3.
    WRITE: / 'Some other error occurred ', SY-MSGV1.
  WHEN OTHERS.
    WRITE: / 'Something is wrong if we get here'.
ENDCASE.
```

The entry point in the C# method is the method with the function module name being called from the SAP system (for example, RFC_CUSTOMER_GET). In Microsoft Visual Studio, you can set a breakpoint here and examine the input values from the SAP system. This provides a similar idea to the ABAP_DEBUG functionality that is provided in the client proxy.

3.3 Monitoring and Debugging SAPServer Stubs

In RFC server code you can set a breakpoint directly in the C# class and see when the function is called by the SAP system. You can monitor TRFC calls to your server from the TRFC monitor in the SAP system with transaction SM58. From here you can also resend the function call to your server. Use the SAP queue monitor (transaction SMQ2) to monitor QRFC calls.

You can do advanced tracing within the SAP system, for example from transaction code SM50 and by examining the work process trace files (for example, dev_w0 in the SAP work directory).

3.4 SAPServer and TRFC

An RFC server application allows you to use .NET functionality within your SAP system. RFC servers can be (normal) RFC, TRFC or QRFC servers.

To write a QRFC server, you have to implement the SAP queuing mechanism. For purposes where you want queuing, we recommend to use *Microsoft Message Queue* in a TRFC server.

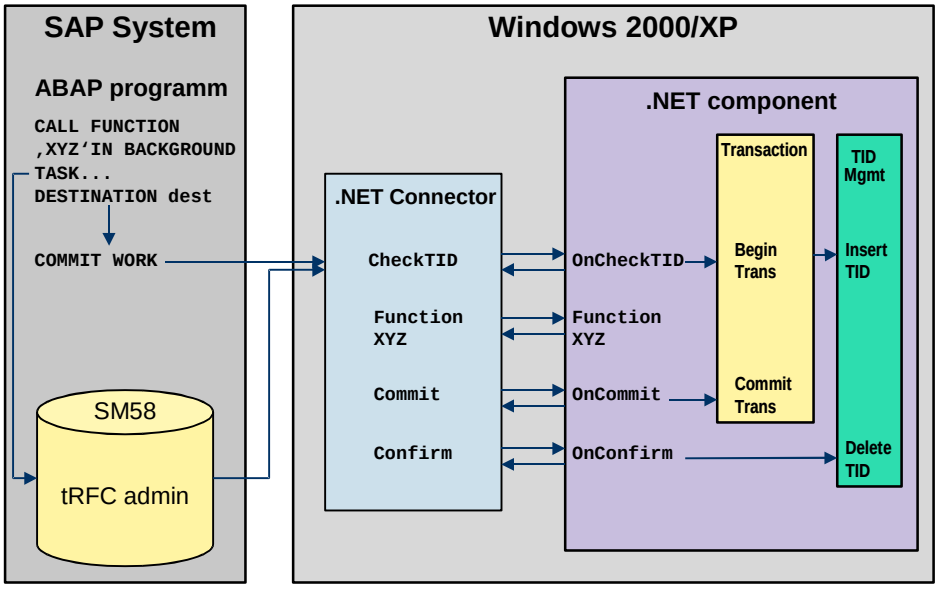
A TRFC server makes sense when have to send information only once from the SAP system to another application (for example, sending a purchase order) . TRFC is required if you want to write an application to receive SAP IDOCS. On your TRFC server, you must manage a connection to a transactional store such as Microsoft SQL Server. You require a transaction store to ensure you can keep track of and manage all `RfcTID` sent to you from the SAP system so that you can create the `TID` and the function execution within a transaction. You should therefore override the base class `CheckTransaction` and `CommitTransaction` methods. The base method returns `0` indicating success.

For TRFC, you must implement the following methods:

	Method	Explanation
1.	`CheckTransaction`	When this method is called you should search your `TID` database to determine if this transaction ID exists or not. If it exists it means that we have already received the request to execute the function. If not, this is the first time and we should log the `TID` with a status that indicates the method is not yet executed. It should return the following: 0 – It is a new `TID`. Begin transaction and insert `TID` into our database 1 – `TID` already exists in our database, but is not yet confirmed in SAP system (client). SAP will confirm on its end. No further action is required Other – there is an error (for example database connection is down. This tells the SAP system to try again later)
2.	`Method itself (void)`	In the transaction, execute the function. If there is a problem, throw an exception
3.	`CommitTransaction`	This method is called if you do not throw an exception during the actual method execution (step 2). When the method is called, you should commit the transaction and return the following: 0 – committed successfully Other – Failure
4.	`RollbackTransaction`	This function is called after the method execution failed because you raised an exception. You can rollback the transaction at this point or in the method.
5.	`ConfirmTransaction`	This method is simply an opportunity for you to clean up your `TID` database by removing `TID` values that have been fully executed 0 – means it is ok Other values – means it is not ok.

If the transaction fails at any point, the SAP system tries to resubmit it. By default, the system attempts to resubmit the transaction every 15 minutes up to a maximum of 30 attempts. However, you can configure the resubmission parameters individually for each RFC destination using transaction SM59. From the destination maintenance screen, choose *Destination → TRFC options.*

Graphically this process looks like this:



3.5 RFC Server Exceptions

If an exception is thrown while the SAPServer is executing, this exception will be passed to the SAPServerHost on which the SAPServer instance is hosted by calling the SAPServerHost's OnServerException function. This is a virtual function and can be overridden in the derived class.

SAP ABAP exceptions can be returned from your RFC Server component by throwing an RFCAbapException. The RfcAbapException contains two strings: error code and message.

Error code is the name of the exception in ABAP/4 (for example, NOTHING_SPECIFIED) and is referenced inside the ABAP/4 program as a SY-SUBRC code. Message is mapped to SY-MSGV1 and can be examined for additional detail.

Example:

```
// exceptions are returned to SAP ABAP/4 program as appropriate SY-
SUBRC & SY-MSGV1
if (""== Kunnr & "" == Name1)
{
RfcAbapException ns = new RfcAbapException("NOTHING_SPECIFIED", "
Both kunnr and name1 were empty");
throw ns;
}
if ("RHD" == Name1)
{
RfcAbapException nrf = new RfcAbapException("NO_RECORD_FOUND", " No
customer by name: " + Name1);
throw nrf;
}
```

For a complete RFC Server example with exception handling, see the RFCServerConsole sample.

3.6 SAPIDocReceiver

The SAPIDocReceiver class is a TRFC server implementation used to receive SAP intermediate documents from an SAP system. An example of an IDOC might be a customer sales order or a material master record. IDOC records are typically used in EDI scenarios. See the IdocReceiverService sample for example code.

```
[C#]
public class SAPIDocReceiver : SAP.Connector.SAPServer
```

Public Constructors

public SAPIDocReceiver (string[] args , SAP.Connector.SAPServerHost host)	Creates a new SAPIDocReceiver and attaches to a SAPServerhost. Gets the connection information from args.
public SAPIDocReceiver (string[] args)	Creates a new SAPIDocReceiver and does not attach to a SAPServerhost. Gets the connection information from args.
public SAPIDocReceiver (System.String ConnectionString , SAP.Connector.SAPServerHost host)	Creates a new SAPIDocReceiver and attaches to a SAPServerhost. Gets the connection information from a connection string.
public SAPIDocReceiver (System.String ConnectionString)	Creates a new SAPIDocReceiver and does not attach to a SAPServerhost. Gets the connection information from args.
public SAPIDocReceiver ( )	Creates a new SAPIDocReceiver without connection information. Does not attach to a SAPServerhost.

Public Events

public event SAP.Connector.SAPIDocReceiver.ReceiveEventHandler BeginReceive	Raised when the IDOC transmission from SAP is first received. You must define an event handler for this event in your code.
public event SAP.Connector.SAPIDocReceiver.ReceiveEventHandler EndReceive	Called at the end of the transmission. You must define an event handler for this event in your code.

Remarks

SAPIDocReceiver is based on SAPServer and can be managed just like other SAPServer classes (added to hosts, started and stopped, etc.) The SAPIDocReceiver adds the BeginReceive and EndReceive events to the base implementation. These events are called when an IDOC transmission is first received and after the transmission is completed respectively. The complexity of working with IDOCs is in large part managed by the component's internal implementation details.

Example

For a complete example see the sample IdocReceiverService.

Creating the variables for IDOC receiver and registering the event handlers

```
private SAPServerHost host;

private SAP.Connector.SAPIDocReceiver idocrec;

private System.IO.StreamWriter sw;

SAPIDocReceiver idocrec = new SAPIDocReceiver(args, host);

idocrec.BeginReceive += new
SAPIDocReceiver.ReceiveEventHandler(this.idoc_BeginReceive);

idocrec.EndReceive += new
SAPIDocReceiver.ReceiveEventHandler(this.idoc_EndReceive);
```

Example: BeginReceive event handler

```
private void idoc_BeginReceive(object sender,
SAP.Connector.SAPIDocReceiver.ReceiveEventArgs e)
{
      // this method is called when an idoc is received.
      // we tell the idoc receiver to write to a stream writer
      EventLog.WriteEntry("Idoc Service", "idoc begin receive");
      sw = new System.IO.StreamWriter(@"C:\temp\idocs.txt", true,
System.Text.Encoding.ASCII);

e.WriteTo = sw;
}
```

Example: EndReceive event handler

```
private void idoc_EndReceive(object sender,
SAP.Connector.SAPIDocReceiver.ReceiveEventArgs e)
{
// this method is called when idoc is done being received
// we can close the stream writer now
EventLog.WriteEntry("Idoc Service", "idoc end receive");
sw.Close();
}
```

4 Data Type Reference

4.1 RFC To .NET Data Type Mapping

Simple Data Types

The following data types are mapped directly to .NET base data types in the main C# proxy class.

ABAP Type	.NET CLS
C (String)	String
I (integer)	Int32
F (Float)	Double
D (Date)	String
T (Time)	String
P (BCD Packed, Currency, Decimal, Qty)	Decimal
N (Numc)	String
X (Binary and Raw)	Byte [ ]
RFC String	String
XString	String

Comments on Simple Data Types

- ABAP type *N* (numeric) data is mapped to a STRING data type in the C# proxy. It contains numeric data such as invoice numbers. Some type *N* fields like invoice number or customer number require you to enter the string with leading zeroes. If in your application the customer number field is numeric, be sure to convert your entry back to the appropriately formatted string before using it in the proxy.
- Date / Time fields are mapped to .NET STRING data types in the C# proxy. The format of the string is the same as the SAP internal storage of the date and time. Specifically for date this is YYYYMMDD and for time it is HHMMSS. The SAP .NET connector provides functions specifically to help you convert from SAP Date/Time fields stored as string to a .NET date or time field.

Complex Data Types

SAP Data Type	.NET Data Type
Structure	C# class derived from SAPStructure
Table (ITAB)	C# class derived from SAPTable
(New) Hierarchical Table (type II ITAB)	Not supported in this version

4.2 SAPTable Class

The SAPTable class represents a very common data type in the SAP system and is frequently used to hold the results of the RFC call. It is occasionally used to pass in selection variables to the RFC call. A SAPTable is made of SAPStructures of a single type. For example a SAPTable called RFCFuncTable would be made up of RFCFunc structures (rows).

```
[C#]
public class SAPTable : System.Collections.CollectionBase
```

Public Constructors

protected SAPTable ( )	Creates a new instance of SAP Table. You should use the SAP table control to reference SAP Tables in your code. The SAP table will be created automatically by the connector.

Public Methods

Add (SAPStructure)	Adds a new row (SAPStructure) to the table
Contains(SAPStructure)	Returns true if the SAPTable contains that row
CopyTo(SAPStructure[ ],int)	Copies into a SAPStructure array up to the given index
CreateNewRow()	Creates a new blank row in the SAPTable
GetElementType()	Returns the type of the particular SAPStructure (for example, RFCFunc or other SAP structure)
IndexOf(SAPStructure)	Returns the current index ID
Insert(int, SAPStructure)	Inserts at the index given a SAP structure
Remove(SAPStructure)	Removes the SAP structure specified from the table
SortBy(string fieldname, string direction)	The SAP table now supports sorting, for example in a datagrid.
ToADODataTable	Creates a new ADO .NET DataTable from the SAP Table.
FromADODataTable	Creates the SAP Table from an ADO .NET DataTable. Note that the schemas must be IDENTICAL.

Public Properties

This[int]	An indexer to get access to the current SAP structure

Example

See the Winform sample for examples of using the SAPTable class.

4.3 SAPStructure Class

The `SAPStructure` class represents a very common data type in the SAP system and is frequently used to hold the results of the RFC call as part of a `SAPTable` or as a BAPI error return. A structure is made up of several simple data types and can be thought of as a row of a table. The `SAPStructure` is generated for you by the SAP .NET Connector and should not be modified outside of the *SAPConnectorGenerator* tool.

```
[C#]
public class SAPStructure : System.Object
```

Public Constructors

`protected SAPStructure ( )`	The `SAPstructure` is the base class for a specific type of SAP Structure. For example a structure containing customer address data (customer, city, state, phone, etc.)

Example

See the Winform sample application for example code dealing with SAP Structures.

4.4 RFC Parameter Mapping to C#

RFC parameters fall into these categories:

- `IMPORT`

  Always pass values from the calling program to the function module unless this is marked as an optional parameter.

- `CHANGING`

  These are passed to the function module from the calling program and are then passed back to the program.

- `EXPORT`

  These are passed from the function module to the calling program unless marked as an optional parameter.

- `TABLES`

  These represent an array of SAP structure instances. They can be `IN` or `OUT` parameters.

- `IMPORT, EXPORT, CHANGING`

  These parameter types cannot be tables. They can be structures or other simple data types.

- `OPTIONAL`

  Parameters can also be defined as optional.

A simple mapping of RFC parameter types to C# parameter types follows:

RFC Parameter Type	C# Parameter Type
`IMPORT`	`IN`
`EXPORT`	`OUT`
`CHANGING`	`REF (In/Out)`
`TABLES`	`REF (In/Out)`

5 Samples

Several samples are provided with the connector to help you understand how the proxies work and to illustrate the topics discussed in this reference. They are provided without warranty or support.

The samples are designed to work against the *Mini SAP Web Application Server 6.20* or any system supporting `RFC_CUSTOMER_GET` and `RFC_CUSTOMER_UPDATE`. For the IDOC samples you need a system than can accept IDOCs. An `Exchange_rate01` IDOC type is provided as a sample that should work with the mini SAP system. All samples are written in Microsoft C#, except for the Visual Basic windows form sample. Before using the samples, be sure to verify the login parameters in the SAP destination component. The samples are in the folder `Samples`. To launch the samples in Visual Studio .NET double-click on `Samples.sln`.

5.1 Windows Form Sample

This sample is a Windows form and illustrates the following concepts:

- SAPClient programming
- Dealing with exceptions
- Asynchronous method calls
- Debugging and tracing
- Working with SAP Tables (sorting, converting to ADO .NET, data binding)
- Synchronous, transactional and Queued RFC updates

The Windows form sample uses the `RFC_CUSTOMER_GET` function module. It accepts as input a customer name string (for example, `A*`) and returns in a datagrid all customers that match that selection.

To use the sample, you have to enter an SAP customer search selection criteria. All functions are available from the sample's application menu.

To use the option *Save Results to SQL Server* you must set the SQL logon parameters in the component `SqlConnection1 on form1`. To do this, click on the property `ConnectionString` and use the drop down box. Let the Wizard construct the connection string for you. You also need a table named `cust` in the Northwind database, and a stored procedure called `Insert_cust`. The SQL script to create both of these objects is located in the subfolder `SQL`.

5.2 Webform Sample

The web form sample is the simple ASP .NET web form shown in the guide and illustrates the following:

- SAPClient programming
- Using ASP .NET with the connector
- Databinding in web form
- Use of the SAPlogin form

Before using this sample, you have to share the folder `DNCWebApp`.

5.3 Simple RFC Server

This console application is a simple RFC server implementing the SAP function `RFC_CUSTOMER_GET`. Our .NET implementation, returns two customers back to the SAP system and prints out the parameters sent to us by the SAP call.

This sample illustrates:

- Use of a Windows Console application with the connector
- A simple RFC Server
- Calling a .NET server from SAP
- Command line parameters for connecting to SAP gateway
- Using the RFC Server host with the same connection information for each server

This sample uses command line arguments to connect to the SAP gateway (for example, `–aMYPROGID –gLOCALHOST –xSAPGW00`), where `–a` is the program ID used in your TCP/IP destination, `-g` is the gateway host and `–x` is the gateway service.

You can set the command line arguments in the Visual Studio .NET debugger by right-clicking on the project `RFCServerConsole` and selecting *Properties* → *Configuration properties* → *Debugging* → *Command Line Arguments*.

Before calling this RFC server in the SAP system, you must have setup a TCP/IP destination in your SAP system (type registration). You can call this sample RFC server by running `RFC_CUSTOMER_GET` in single test from the RFC function builder (transaction `SE37`) inside of the SAP system, with the destination of your TCP/IP destination. Alternatively, you can write an Abap/4 program as described in Calling our RFC .NET Server from SAP Programs [Page 70].

An example ABAP/4 program is provided in this sample's "ABAP" directory.

5.4 IDOC Receiver as a Windows Service

The windows service sample takes the SAP IDOC Receiver component and deploys it as a Windows Service. You can see:

- How RFC servers and Windows Services are similar
- How to install a Windows Service that is also an SAP RFC .NET server
- Basic functions of the IDOC Receiver
- How to manage the connection parameters in the Windows Service properties

Before running this sample, you must install it using the Visual Studio utility `Installutil.exe`.

Using the Visual Studio .NET command line, navigate to the folder containing `Installutil IdocReceiverService.exe` (for example, `SAP .NET Connector\Samples\IdocReceiverService\bin\Debug`).

Run the command `Installutil IdocReceiverService.exe`. This should install the Windows Service. You can verify this by looking at the Services manager in the Administrative Tools. The Service name is `.Net connector Idoc Service`.

To uninstall the Windows service run the command `Installutil -u IdocReceiverService.exe`.

You set the SAP connection string in this service's `Start parameters` property. For example if your TCP/IP destination used the program ID of `myProgID` and your SAP system ran on your local machine, then the start parameters would be:

`-amyProgID –glocalhost –xSAPGW00`.

Messages are logged to the *Windows Application Event Viewer*. When received, SAP IDOCs are appended to the following file: `C:\temp\idoc.txt`.

To enable your system to send IDOCS you must have a TCP/IP destination (type registration), a configured partner profile (`WE20`) and a configured TRFC Port (`WE21`). The *Mini SAP Web Application Server 6.20* supports the IDOC type `EXCHANGE_RATE01`. For convenience, you can find a sample IDOC file in the IDOC submitter sample's `SampleIdoc` folder. If you have no IDOCs on your system you can submit this IDOC using the `IdocSubmit` sample application. Then use the IDOC test utility (`WE19`), change the IDOC header and submit it as an outbound IDOC to the TRFC Port representing this `IdocReceiver` sample.

5.5 IDOC Submitter Windows Form

This application is a Windows form that loads an IDOC file and displays it on the form. You can then submit the IDOC using the .NET connector's `IdocSender` component. For convenience, an `EXCHANGE_RATE01 IDOC` (ANSI encoded) file is provided in the folder "`SampleIdoc`". This sample illustrates the following:

- Sending an IDOC file to SAP system
- Creating and confirming SAP Transaction ID (TID)
- Working with a stream reader to open IDOC file and display on Windows form.

5.6 Simple RFC Web Service

This application is an ASP .NET web service interface to `RFC_CUSTOMER_GET`. The web method signature can be the same method signature as the SAP RFC method or it can be a simplified method signature. However, since SAP Tables are complex data types you must write a client to execute the web method. In other words, you cannot run the web service using the default Web Services test harness provided by Visual Studio .NET. To allow you to run the web service with the least effort, the method signature was slightly modified.

Before running this application you have to share the folder `DNCWebServiceSample`.

This sample illustrates a simple web service wrapper for `RFC_CUSTOMER_GET`.

5.7 Simple Visual Basic Windows Form

The VBWinform project shows how to use a compiled SAP Proxy which was generated in C# within your Visual Basic .NET project. To use Visual Basic or other CLR language with the connector you simply have to set a reference to an existing or a new proxy. The proxy could be in the global assembly cache (recommended) or within the same solution as your VB.NET project (shown in this sample).

5.8 X.509 Certificate Sample

Before using this sample, share the folder `\SAP .NET Connector\Samples\DNCX509Cert\`. This sample shows how to use X.509 certificates with the connector. Note that you must run this sample in HTTPS to get the X.509 certificate from IIS.