

SQL Server CE Database Development with the .NET Compact Framework

ROB TIFFANY

SQL Server CE Database Development with the .NET Compact Framework
Copyright ©2003 by Rob Tiffany

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-59059-119-4

Printed and bound in the United States of America 12345678910

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Technical Reviewer: Darren Flatt

Editorial Board: Dan Appleman, Craig Berry, Gary Cornell, Tony Davis, Steven Rycroft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Nate McFadden

Copy Editor: Ami Knox

Production Manager: Kari Brooks

Production Editor: Lori Bring

Proofreader: Lori Bring

Compositor: Argosy Publishing

Indexer: Ron Strauss

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States, phone 1-800-SPRINGER, email orders@springer-ny.com, or visit <http://www.springer-ny.com>. Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit <http://www.springer.de>.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit <http://www.apress.com>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.

Remote Data Access

NOW YOU'RE IN the home stretch. You've learned about the inner workings of SQL Server CE and how to program against it with ADO.NET. These final two chapters will take you outside the database and show you how to securely move data back and forth between SQL Server and SQL Server CE. Not only that, you'll be able to add new data and change existing data on your Pocket PC and see those changes and additions reflected on the server. At the beginning of the book, I discussed some things that you don't usually see in a computer book. Remember those FBI agents working in Phoenix and Minneapolis who couldn't get their critical field memos to the right people in Washington, D.C. before 9/11? Well, in this chapter and the next, you're going to build a system that will facilitate the creation and movement of simple field memos between a Pocket PC and a SQL Server. I'm not a business intelligence expert, so I'll leave it to someone more qualified than myself to implement a data mining system to analyze these memos in order to connect the dots. In this chapter, I'm going to cover doing this with a technology called Remote Data Access, or RDA for short. Compared to the next chapter's coverage of merge replication, RDA is relatively easy to get up and running and works with SQL Server versions 6.5, 7.0, and 2000.

RDA Architecture

RDA provides a framework of client-side and server-side components that allow devices running SQL Server CE to communicate with SQL Server 6.5/7.0/2000 via HTTP and HTTPS. Internet Information Server (IIS) acts as the intermediary in the transfer of data back and forth between the Pocket PC and the server. Due to the use of IIS, RDA can take advantage of things like the ability to utilize proxy servers and communicate through firewalls, and it can make use of various security mechanisms. RDA is also well suited for wireless or slower-speed networks due to its use of data compression.

Client Agent

The SQL Server CE Client Agent runs on your Pocket PC and sits in between your .NET Compact Framework application and the SQL Server CE Database Engine. It exposes all the properties and methods that your application will use to control RDA. Based on the methods you call, the Client Agent will manipulate the Database Engine and communicate with the SQL Server CE Server Agent via IIS and HTTP/S. The three methods included in the `SqlCeRemoteDataAccess` class that do all the work for your RDA application are Pull, Push, and SubmitSQL.

Pull

The Pull method is designed to retrieve tables, indexes, and data from SQL Server and store it locally in a SQL Server CE database. You can optionally choose to have the local inserts, updates, and deletes of this retrieved data tracked by the Database Engine.

Push

The Push method is designed to take any tracked local data and send the changes to the Server Agent, where it will execute the appropriate inserts, updates, or deletes against SQL Server.

SubmitSQL

The SubmitSQL method allows you to submit SQL statements for execution on the remote SQL Server. Only inserts, updates, deletes, and calls to stored procedures are allowed, since this method isn't designed to return rows to the Pocket PC.

Database Engine

The SQL Server CE Database Engine is designed to manage the data store on your Pocket PC. When the Pull method is invoked, the Database Engine will create the appropriate tables and indexes as necessary. The Database Engine even tracks the database records that are inserted, updated, and deleted when the Pull method is called with the tracking option enabled.

Server Agent

The SQL Server CE Server Agent runs as an ISAPI DLL on your IIS server, where it handles HTTP/S requests made by the Client Agent and then makes calls to the SQL Server OLE DB Provider to fulfill those requests. The ISAPI extension is called `sscesa20.dll`, and it resides in the special IIS virtual directory that you create for your RDA application.

Security Considerations

Normally, you only have to deal with one set of security credentials when working with an application. Both RDA and merge replication require you to work with two different security schemes. IIS provides the gateway to moving your data back and forth across the Internet, so you have to deal with its security first. SQL Server has its own security requirements that you have to meet to access the data you need to get the job done. Both IIS and SQL Server utilize Windows security behind the scenes, so technically you've got three security hurdles to overcome. When you mesh these systems together, you're faced with myriad technical challenges.

IIS Security

IIS supports Anonymous Access, Basic Authentication, and Integrated Windows Authentication for gaining access to its resources. Additionally, IIS provides support for Secure Sockets Layer (SSL) encryption in order to protect your data stream as it flows across the Internet. These different security options are described in Table 9-1.

Table 9-1. IIS Security Options

AUTHENTICATION	DESCRIPTION
Anonymous Access	This basically means that there is no security. Anyone can access the server without being authenticated.
Basic Authentication	Access to server resources requires a valid username and password. These credentials are sent over the wire using Base64 encoding, which is simply a means of encoding binary data as a string, and doesn't involve encrypting the data. Therefore, this form of authentication must be combined with SSL to keep both the security credentials and the data safe.

Table 9-1. IIS Security Options (Continued)

AUTHENTICATION	DESCRIPTION
Integrated Windows Authentication	Server resources are accessed via the use of a Windows Domain account. A hashing algorithm is used to encode the username and password as they're sent over the wire. This method of authentication will only work if the Pocket PC, IIS, and SQL Server are on the same LAN and are all part of the same Windows Domain. Furthermore, this form of authentication won't work through firewalls or proxy servers, nor will it work if your SQL Server is running on a different computer from IIS. These limitations will go away with the release of Pocket PC 2003 due to its support for Kerberos.
Secure Sockets Layer (SSL)	All data sent over the wire is encrypted at levels all the way up to 128-bit. The client and the server establish a trust relationship based on a Certificate Authority like VeriSign.

SQL Server Security

SQL Server supports Integrated Windows Authentication as well as SQL Server Authentication for gaining access to its resources. These two security options are described in Table 9-2.

Table 9-2. SQL Server Security Options

AUTHENTICATION	DESCRIPTION
SQL Server Authentication	This is the same type of database security mechanism used by all major databases, in which an unencrypted username and password are sent over the wire. Therefore, it is recommended that these credentials are passed from server to server on a secure LAN behind a firewall, and not out across the Internet.
Integrated Windows Authentication	This type of authentication requires a trusted Windows Domain username and password that are mapped to a SQL Server username and password.

Conclusions

A number of things are going on here to make all the components of an RDA system come together. As you might imagine, whatever credentials you send to IIS

are being used by SQL Server to gain access to the data you need. No matter what type of IIS authentication you choose, the credentials you send could be mapped to a Windows Domain user account that is then utilized by SQL Server, which is using Integrated Windows Authentication. You know for sure that nobody is going to use Anonymous Access because there is zero security, so go ahead and cross that off your list. You also know that Integrated Windows Authentication can't be delegated across multiple servers and won't work with firewalls or proxy servers either. This means that you should only go with this method of authentication if all your assets are on the same LAN and both your SQL Server database and IIS are running on the same box. I imagine that you would only see this configuration in a development or test scenario but not in production.

Clearly, you're going to use SQL Server Authentication on the database side and you're going to use Basic Authentication along with SSL on the Internet Information Server side of things. In your RDA code, you're going to pass along a username and password to satisfy the Basic Authentication requirements of IIS, and you're going to potentially pass along a different username and password to satisfy SQL Server's internal authentication needs. The reason you're going to do this is for flexibility and scalability. If users of an RDA-like system were in the office and on the LAN, they'd definitely use their desktop PC instead of a Pocket PC to get the message across. Obviously, RDA technology is designed for people working out in the field and not in the office. If you're an agent out and about writing a memo about suspicious activity in Houston, Texas, you're probably not on a Windows Domain. When it comes to scalability, the thought of running SQL Server and IIS on the same server is abhorrent. If you have thousands of people working all over the world who need to routinely sync up with the home office, you're going to have to configure your servers to handle that load. You'll need to create highly scalable sync-farms utilizing load-balanced servers running IIS. This gives you the flexibility to scale out when necessary to keep up with the demands of more and more simultaneously connected users. Furthermore, you may want to take advantage of SQL Server's clustering capabilities on the data tier to help beef up throughput there as well. So remember: Basic Authentication, SSL, SQL Server Authentication, and separate SQL Server and IIS servers.

Installing the Server Tools

Before anything can happen with this RDA project, you must first install the SQL Server CE Server Tools. Since you are likely working with SQL Server 2000 by now, it's a sure bet that you've installed Service Pack 3 on your database server in order to protect it from the Slammer worm. You can determine which version of SQL Server you're running by executing @@version in Query Analyzer. Therefore, you'll need to download the latest version of the Server Tools that is designed to work

with SQL Server 2000 Service Pack 3 at <http://www.microsoft.com/sql/downloads/ce/sp3.asp>. The actual file that you'll be downloading is `sqlce20sql2ksp3.exe`. Once you've downloaded this file, install the Server Tools on the computer that's running IIS. The requirements for this server are described in Table 9-3.

Table 9-3. Server Tools Server Requirements

OPERATING SYSTEM	WEB SERVER	SERVICE PACK
Windows NT	IIS 4	SP6
Windows 2000	IIS 5	SP3
Windows XP	IIS 5.1	SP1
Windows Server 2003	IIS 6	

I've shown the first screen of the SQL Server CE Server Tools Setup Wizard in Figure 9-1 just so you can make sure you're installing the correct product.

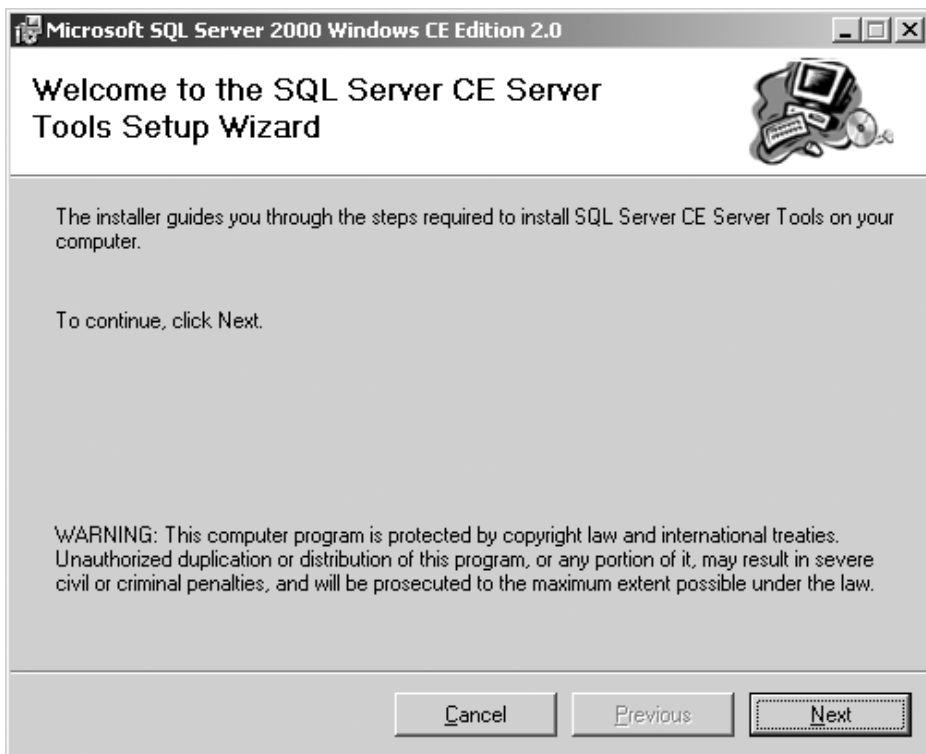


Figure 9-1. SQL Server CE Server Tools Setup Wizard

After you've made it through the setup wizard and you arrive at the final screen as shown in Figure 9-2, make sure that the checkbox to launch the SQL Server CE Virtual Directory Creation Wizard is checked before you click Close.

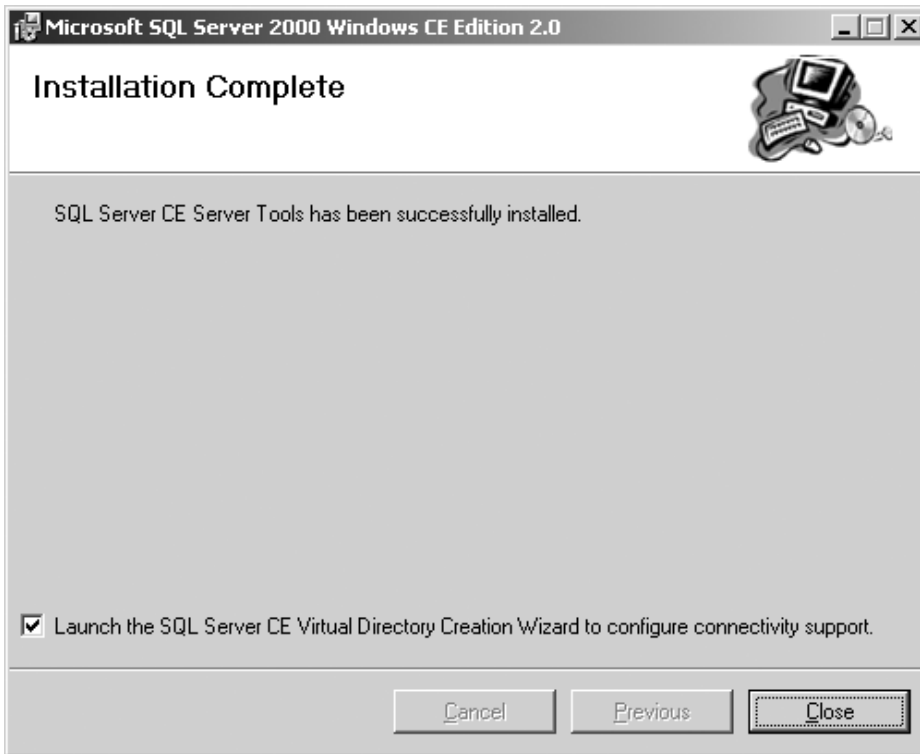


Figure 9-2. Installation complete

When working with the server-side components of SQL Server CE in the past, it was usually up to you to configure the IIS virtual directory and set up the appropriate HTTP and NTFS permissions. All this is now a thing of the past with the advent of the SQL Server CE Virtual Directory Creation Wizard as shown in Figure 9-3.

The first screen after starting this wizard creates a virtual directory as shown in Figure 9-4. You're prompted to enter an alias for this virtual directory, and then you need to associate a new or existing folder with the virtual directory. The folder will contain a log file and a file called `sscesa20.dll`, which is the SQL Server CE Server Agent. For the purposes of this RDA project, enter `FieldAgentRDA` for the alias and `C:\FieldAgentRDA` for the folder, and then click Next.

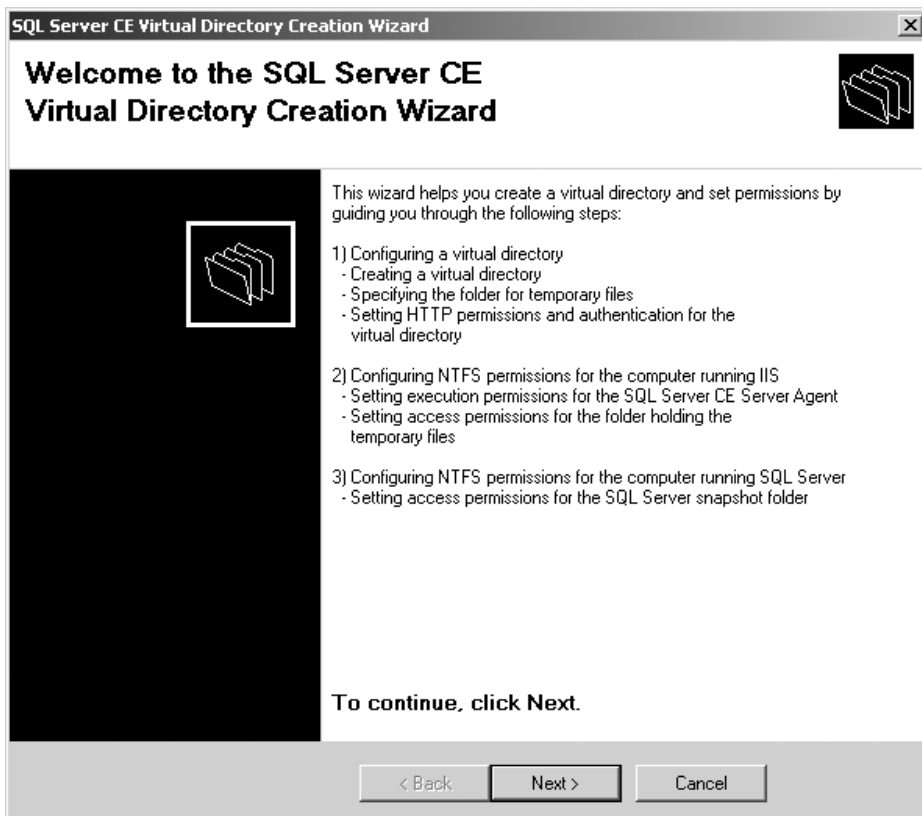


Figure 9-3. SQL Server CE Virtual Directory Creation Wizard

Once you click Next, the dialog box shown in Figure 9-5 will pop up to notify you that the C:\FieldAgentRDA folder you created doesn't contain the SQL Server CE Server Agent. Click Yes and the wizard will automatically copy the Server Agent to your folder and then register it.

The next screen allows you to choose which kind of authentication you want to use for your virtual directory, as shown in Figure 9-6. Your choices are Anonymous Access, Basic Authentication, and Integrated Windows Authentication. You surely don't want to use Anonymous Authentication, because you're building an application that can only be used by valid field agents rather than random people on the Internet. I wouldn't necessarily suggest the use of Integrated Windows Authentication, because it's better suited to an intranet application. Furthermore, Integrated Windows Authentication won't work for you in a multiserver scenario if you decide to use merge replication. The only option is Basic Authentication, through which you'll be sending your credentials over the Internet via clear text. Not to worry, you'll be protecting your data with SSL. You might notice that underneath the Basic authentication checkbox, the screen says Domain: along with

either your computer's name or the domain that it's logged into. What this means is that when it's time to pass in your username from your Pocket PC, you won't have to pass in the domain name in front of it, since it already knows which domain you're coming from.

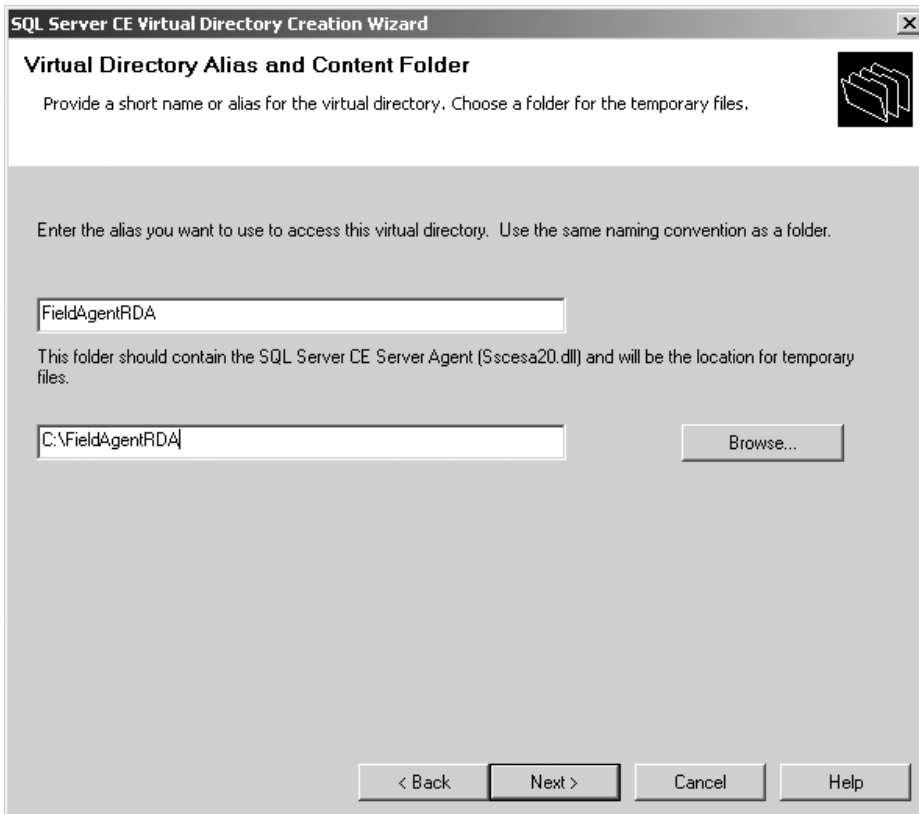


Figure 9-4. Creating a virtual directory alias and content folder

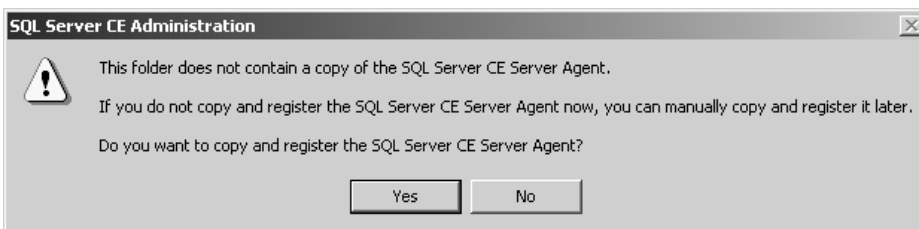


Figure 9-5. Copying and registering the SQL Server CE Server Agent

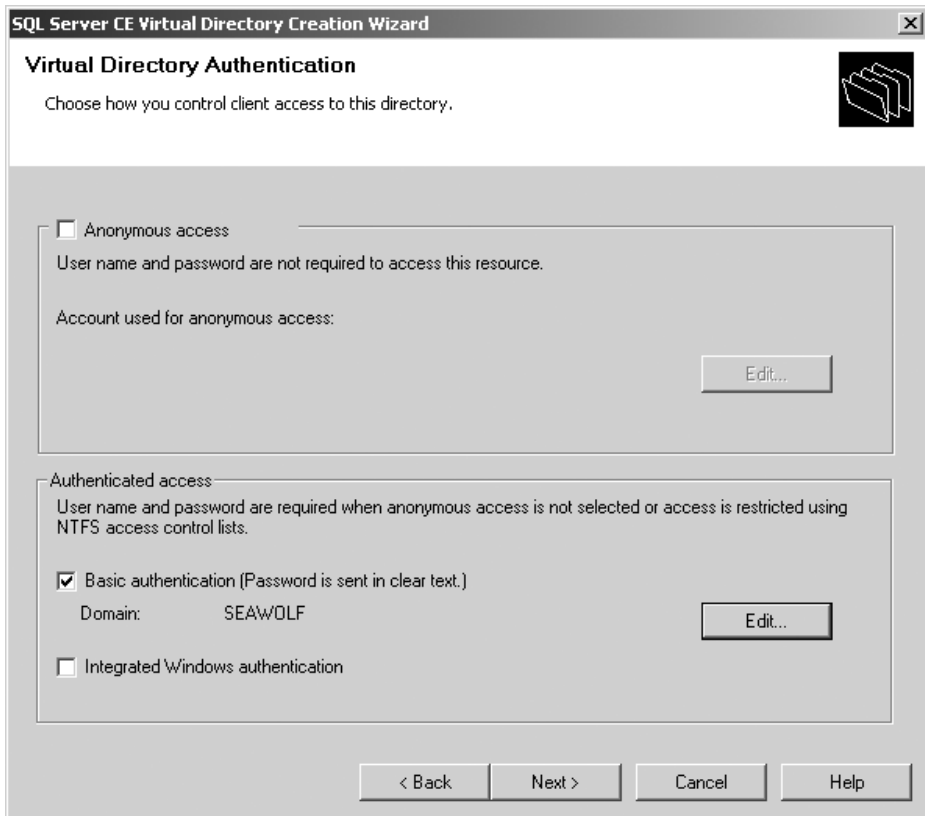


Figure 9-6. Virtual directory authentication

Now that you've set up your HTTP authentication, you must take care of NTFS permission issues on the next screen as shown in Figure 9-7. Despite the fact that the wizard already knows which domain you belong to, on this screen you must enter a valid domain\username. Keep in mind that this username is the same one that each field agent will be passing in as his or her credentials from the field.

You'll disregard the next screen, shown in Figure 9-8, and click Next, as it's designed to set up NTFS permissions for SQL Server snapshot folders when utilizing merge replication.

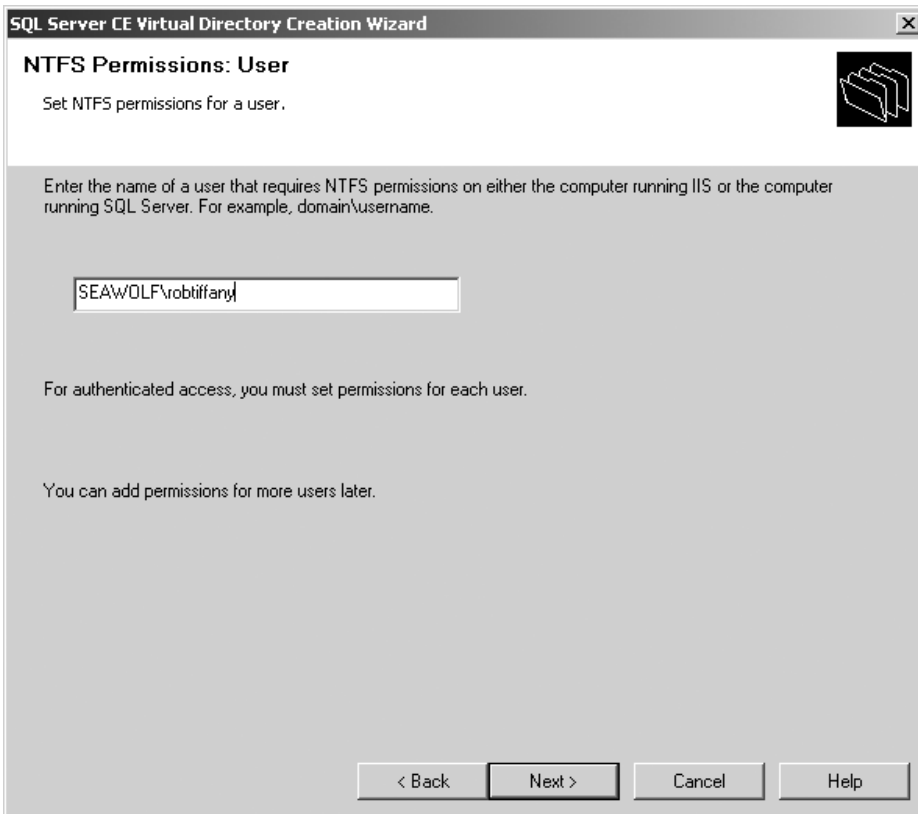


Figure 9-7. NTFS user permissions

The final screen, as shown in Figure 9-9, gives you a quick summary of the choices you made. Make sure and review what the screen displays so that you don't have any unexpected surprises when you make your first attempt to use your RDA application.

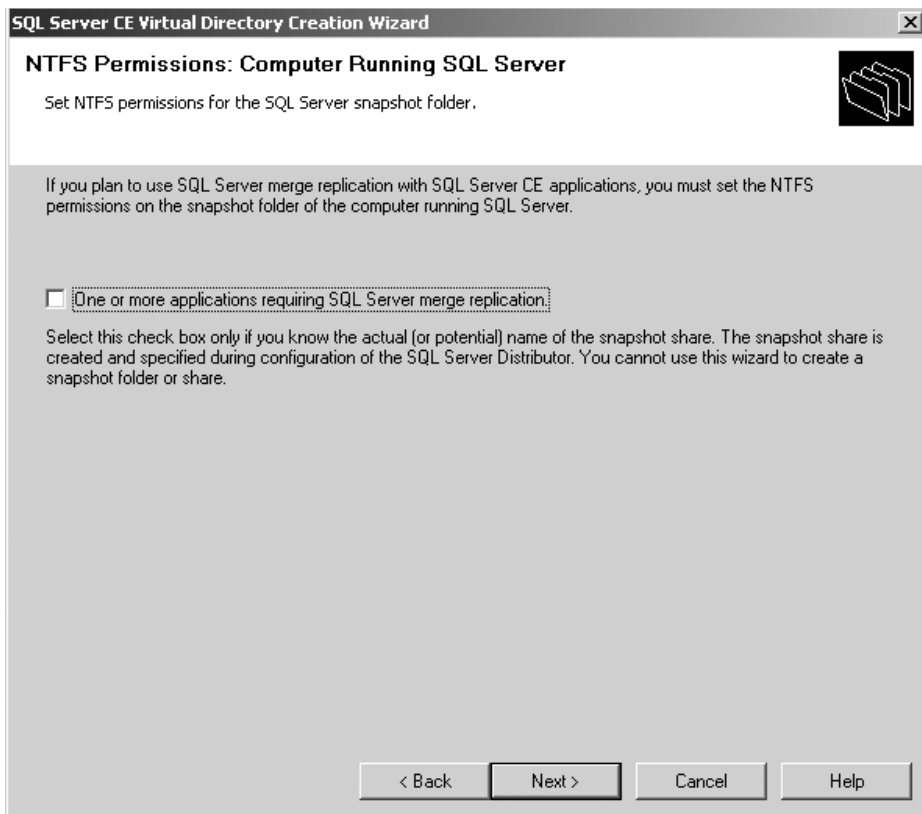


Figure 9-8. NTFS permissions for SQL Server snapshot folders

Now that you've finished the wizard, there are a few things you can do before you execute any code to ensure that everything was set up correctly. First of all, bring up the Windows Explorer on your IIS machine and make sure that a folder called FieldAgentRDA was created in your C:\ drive, if that's where you chose to put it. Next, you need to do the all-important test to make sure that your Pocket PC device can see the Server Agent in your virtual directory. With your Pocket PC connected to your IIS computer via either a network or ActiveSync, bring up Pocket Internet Explorer and point it to `http://<servername or serverIP>/fieldagentrda/sscesa20.dll`. Hopefully, you'll be presented with a screen that looks like the one shown in Figure 9-10.

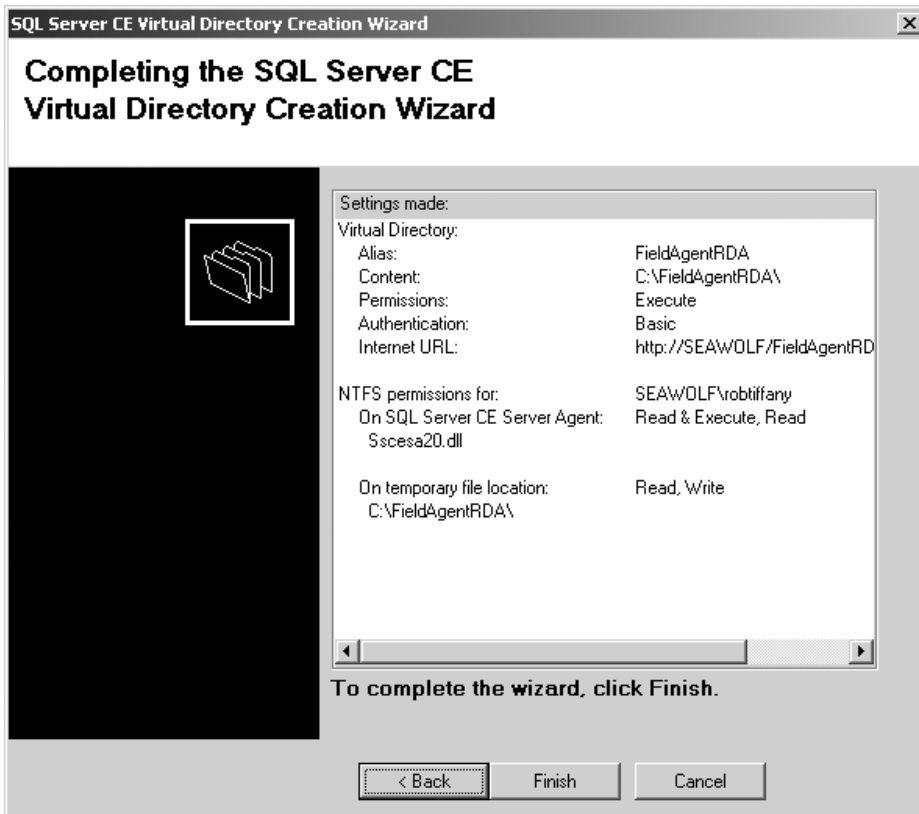


Figure 9-9. Completion of SQL Server CE Virtual Directory Creation Wizard

Remember when you chose Basic Authentication for your virtual directory? This screen is a reflection of that choice since you're being asked to provide a username and password that will be sent to IIS via clear text. If you had chosen Anonymous Access, you wouldn't have been prompted for anything, because IIS wouldn't have cared who you were. The same lack of prompting would have occurred if you chose Integrated Windows Authentication, just as long as you were logged onto a Windows Domain. When you enter the username and password for the user you chose in the wizard, you should be presented with a Web page for SQL Server CE Server Agent, like the one in Figure 9-11.



Figure 9-10. Pocket Internet Explorer prompting for credentials



Figure 9-11. Successfully viewing the Server Agent from Pocket Internet Explorer

Securing Your Connection

You've tested your connection to the Server Agent and everything works great, so you're probably thinking that now it's time to write some RDA code and deploy the application. Wrong answer. Unless you want everyone on the Internet to see your username and password along with the sensitive memos sent in from field agents, you had better do something about encrypting that data stream. The correct answer in this situation is Secure Sockets Layer or SSL. In order to do this, you're going to get a little help from VeriSign, since it allows you to download trial SSL Server IDs for free. Head on over to <http://www.verisign.com> and click the SSL Trial ID link on the home page. At this point you'll be presented with a page asking for some personal information in order to help out their marketing folks. Once you submit that information, you'll be presented with a page that provides you with some info before you get started. Go ahead and click Continue so you can get started generating your Certificate Signing Request or CSR.

Generating a CSR

To illustrate all the steps required to get up and running with your Web server certificate, I'm going to show you how to do so using Windows 2000 and IIS 5.0, since this is currently the most popular combination in the Windows world. I'm sure Windows Server 2003 will take the lead in a few years from now. To get started, navigate to the Administrative Tools on your Start menu and then select Internet Services Manager. Right-click the Default Web Site and select Properties to bring up the Default Web Site Properties dialog box as shown in Figure 9-12.

Select the Directory Security tab and then click the Server Certificate button in order to bring up the Web Server Certificate Wizard as shown in Figure 9-13.

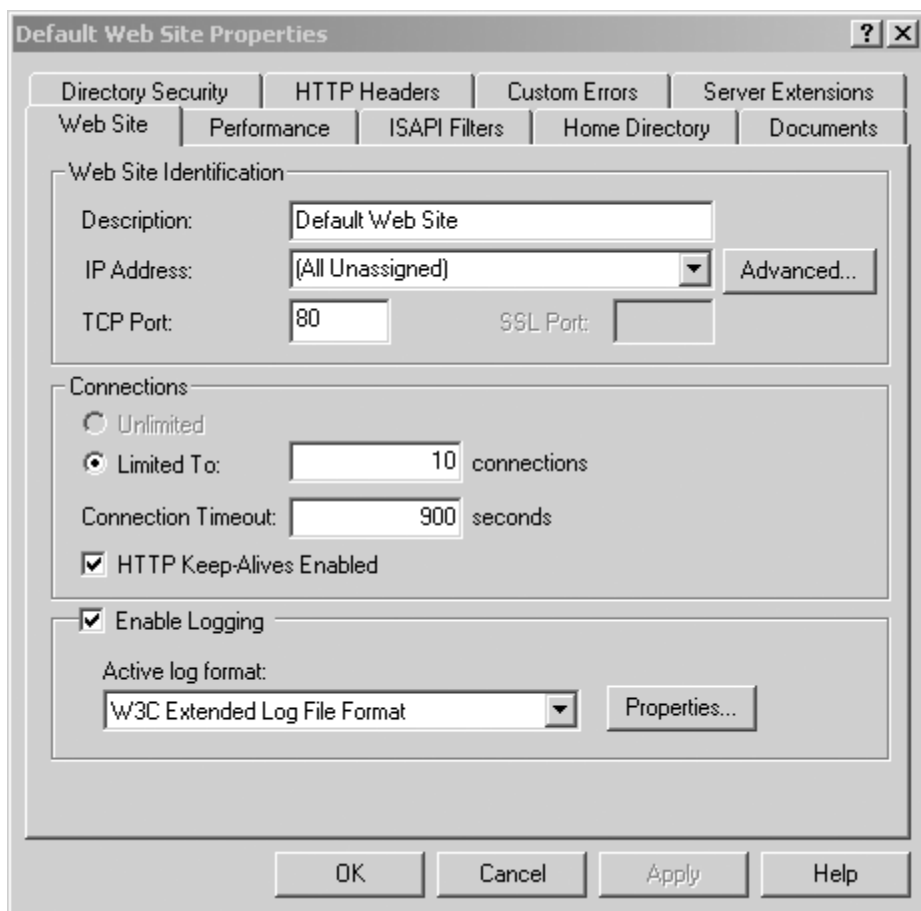


Figure 9-12. Default Web Site Properties dialog box

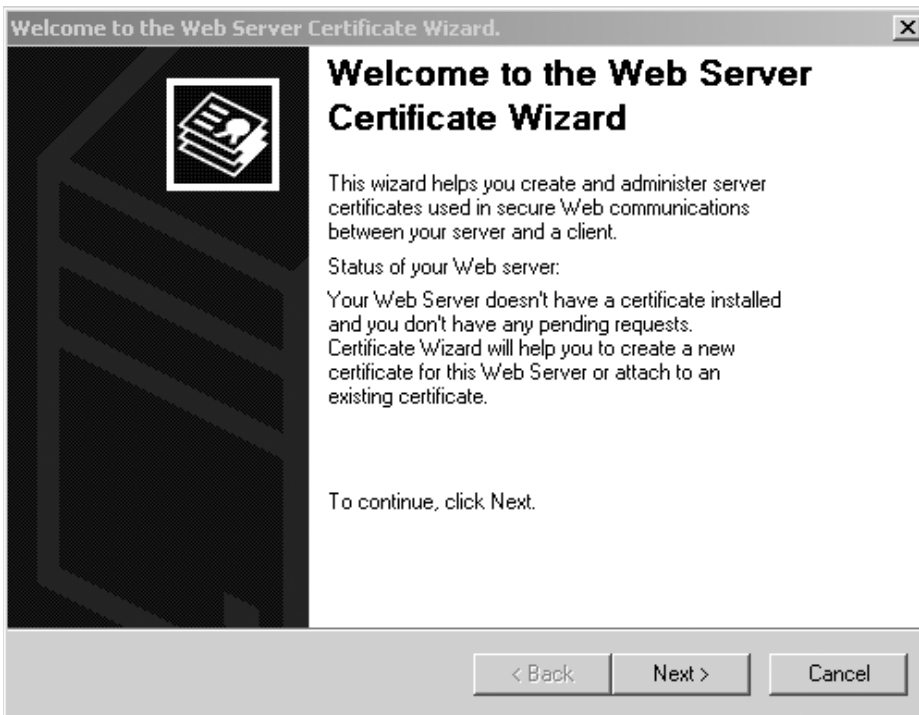


Figure 9-13. Web Server Certificate Wizard

Click Next and then select Create a new certificate as shown in Figure 9-14; now click Next again.

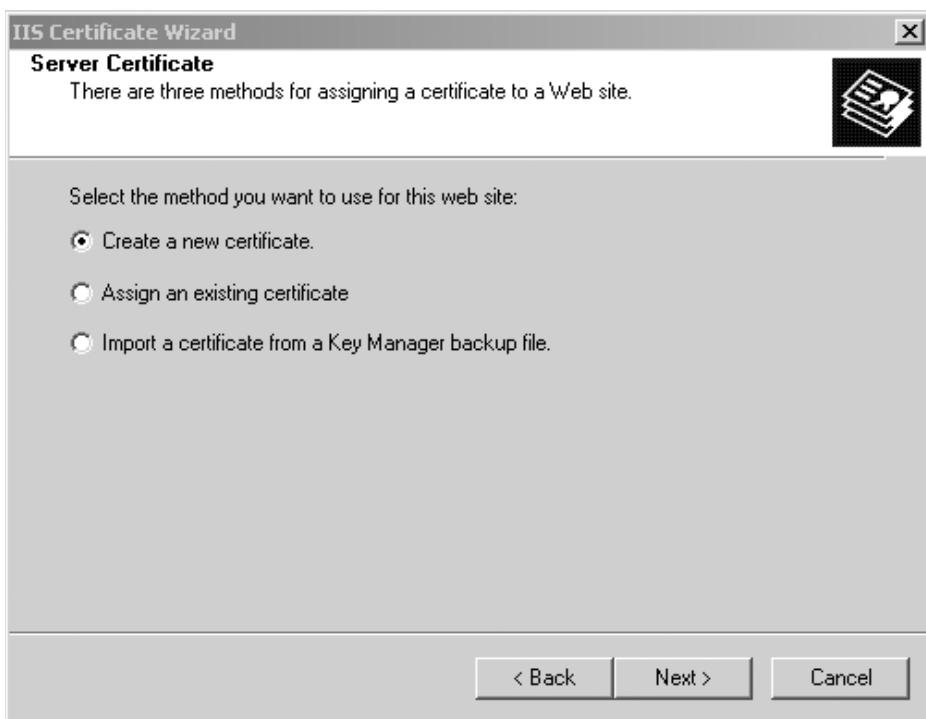


Figure 9-14. Creating a new certificate

On the next screen, ensure that the “Prepare the request now, but send it later” option is selected, as shown in Figure 9-15, and then click Next.

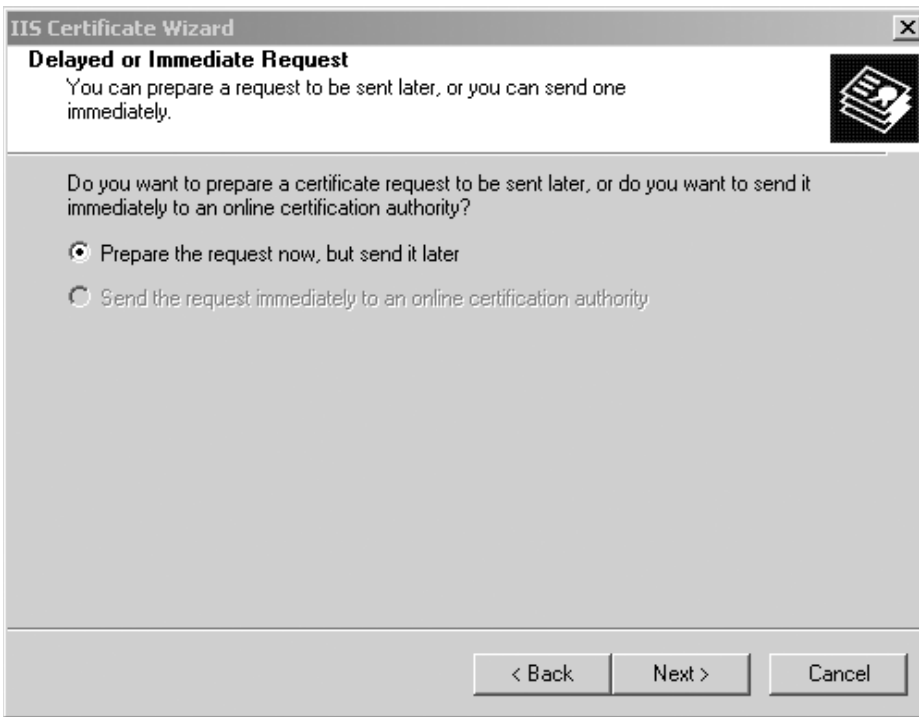


Figure 9-15. Specifying a delayed or immediate request

On this next screen, you get to specify the certificate name and strength as shown in Figure 9-16. Enter FieldAgentRDA as the name and set the bit length to an encryption strength of no more than 1024 so that you don't degrade the performance of your application.



Figure 9-16. Name and security settings

Over the next several screens, you'll begin entering information that will identify you as the owner of the certificate that you're going to generate. On this first screen, you'll need to enter the real or made up name of your organization along with an organizational unit inside your company or organization as shown in Figure 9-17.

IIS Certificate Wizard

Organization Information

Your certificate must include information about your organization that distinguishes it from other organizations.

Select or type your organization's name and your organizational unit. This is typically the legal name of your organization and the name of your division or department.

For further information, consult certification authority's Web site.

Organization:

robtiffany.com

Organizational unit:

Handheld Development Department

< Back Next > Cancel

Figure 9-17. Organization information

On the next screen you're asked to provide a common name for your Web site. Normally, this would be a fully qualified DNS name that includes the name of the host computer. An example might be `server1.microsoft.com`. A more unorthodox approach that works with Windows servers in a Microsoft-based intranet is to enter just the computer's NetBIOS name. For the purposes of this book, that's exactly what I want you to enter. You'll probably be setting up this example on your desktop PC or laptop. The likelihood that you'll be able to get your Pocket PC to resolve your desktop PC via a valid DNS name is low in a situation where ActiveSync and a USB cable are the only providers of your connectivity. Luckily, the Your Site's Common Name screen defaults to your computer's NetBIOS name as shown in Figure 9-18. As you can see, my laptop is called `seawolf`, named such because I used to drive submarines for a living.

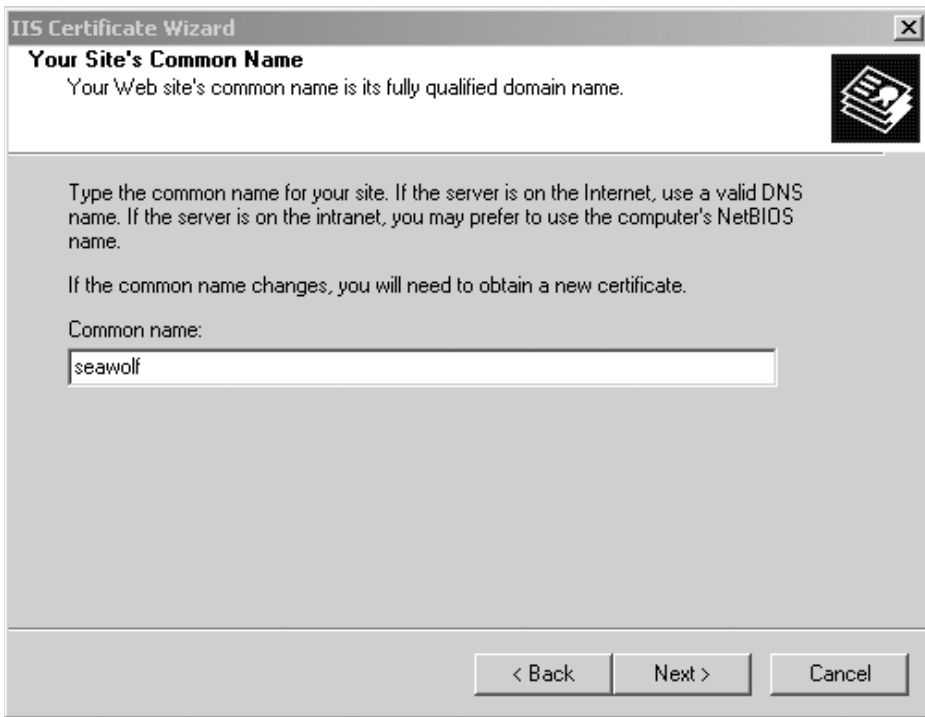
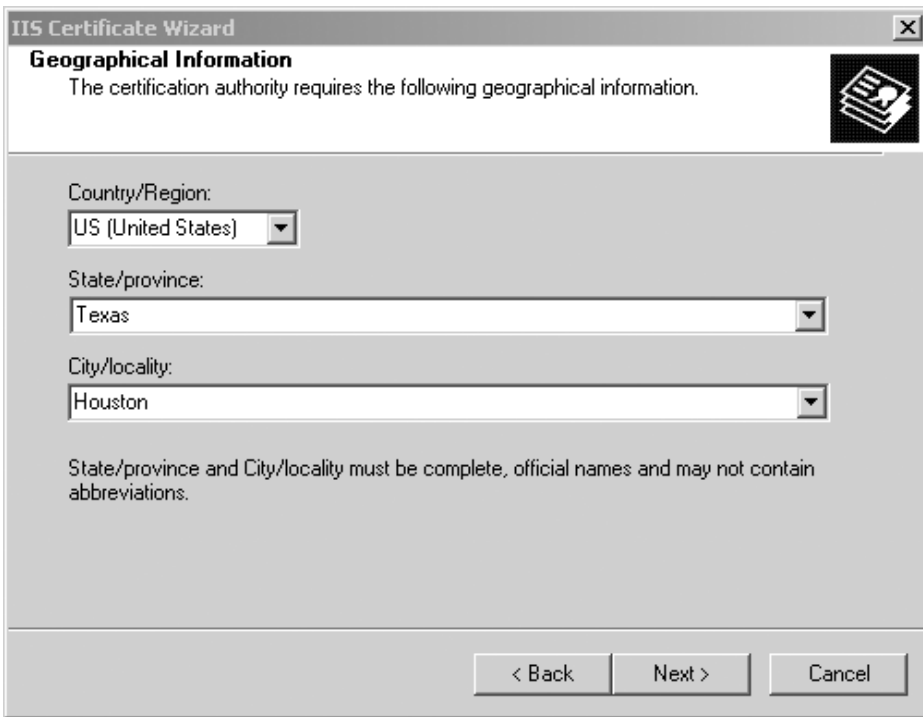


Figure 9-18. Your site's common name

The next screen requires geographical information as shown in Figure 9-19. Enter your country, state, and city and then click Next.



The screenshot shows the 'IIS Certificate Wizard' window, specifically the 'Geographical Information' step. The window has a title bar with 'IIS Certificate Wizard' and a close button. Below the title bar, the section is titled 'Geographical Information' in bold. A message states: 'The certification authority requires the following geographical information.' To the right of this message is an icon of a document with a lightbulb. The form contains three dropdown menus: 'Country/Region:' with 'US (United States)' selected, 'State/province:' with 'Texas' selected, and 'City/locality:' with 'Houston' selected. Below these fields, a note reads: 'State/province and City/locality must be complete, official names and may not contain abbreviations.' At the bottom of the window are three buttons: '< Back', 'Next >', and 'Cancel'.

IIS Certificate Wizard

Geographical Information

The certification authority requires the following geographical information.

Country/Region:
US (United States)

State/province:
Texas

City/locality:
Houston

State/province and City/locality must be complete, official names and may not contain abbreviations.

< Back Next > Cancel

Figure 9-19. Geographical information

On the next screen you'll need to enter a path and filename as shown in Figure 9-20 since your certificate request is saved as a text file.

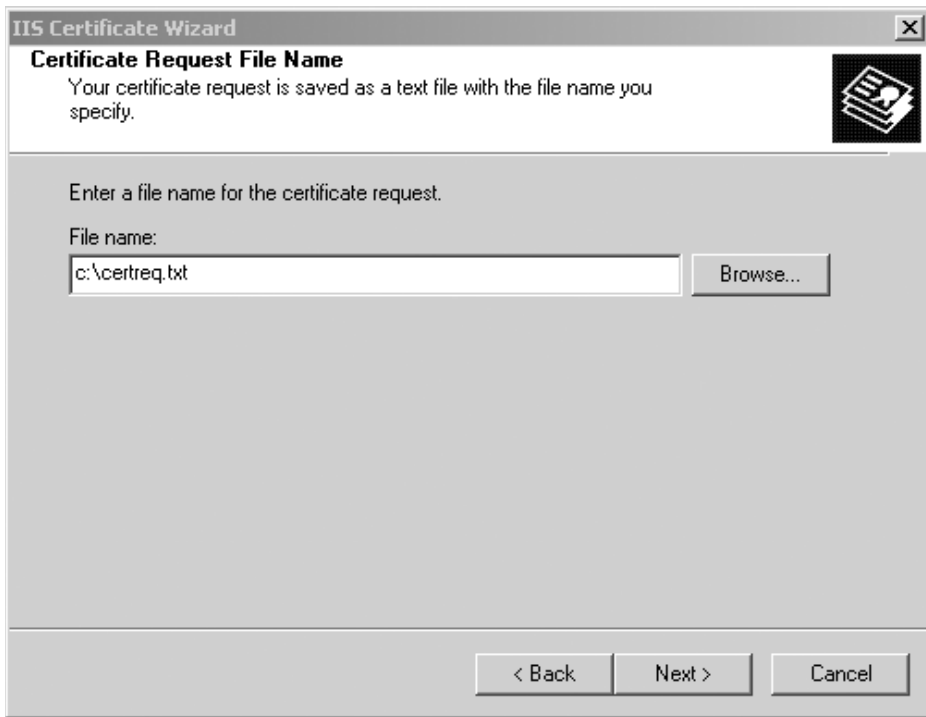


Figure 9-20. Certificate request filename

The next screen provides you with a summary of your choices. Click Next to move on to the final page where you can click Finish—and then you're done.

Submit Your CSR

Click the Continue button on the VeriSign Web site in order to move on to step 2 of 5 through which you'll get to submit your CSR. Navigate to wherever you saved your certificate request text file on your hard drive and open it in Notepad. Choose Select All from the Notepad menu and then select Copy. Now go back to the Web page and paste your information into the text area as shown in Figure 9-21 and then click Continue.



Enrollment

Step 2 of 5: Submit CSR

Before you Start

Step 1: Generate CSR

• Step 2: Submit CSR

Step 3: Complete Application

Step 4: Install Test CA Root

Step 5: Install your Test Server ID

Submit CSR

When you generated the CSR in Step 1: Generate CSR, your server software either e-mailed the CSR to you, or created a request file on your hard disk (such as key.req). Open the CSR file with an ASCII text editor such as Notepad. (Do not use a word processor such as Word that inserts formatting or control characters.)

This is an example CSR file:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBCTCBtAIBADBPMQswCQYDVQQGEwJVUzEQMA4GA1UECBMRmxvcm1kYTEYMBYG
A1UEChMPRX11cyBvbiBUaGUgV2ViMRQwEgYDVQQDFAt3d3cuZXR3Lm5ldDBcMAOG
CSqGSIB3DQEBAAQAA0sAMEgCQQCeojtjnHggDGTxp+XZ56RaSe1iZWpumXjU6Sx7
v1FdXzsY1oLQa090Jtnu1WsQRHhOyDS+45oncjKmlzCG/IZAgMBAAGgADANBgkq
hkiG9w0BAQQFAANBAFBj9g+N1Uh8YWPPrFGntgf4miUd/wqUshptjJy4PjdsD3ugy
5avvuh3G//PpGh2aYXIjHpJXTUBQyzzSEIINYtc=
-----END NEW CERTIFICATE REQUEST-----
```

Description	
Enter CSR Information: Copy the entire contents of the CSR file including the lines that contain the begin and end statements into the field on the right.	<pre>bgBuAGUAbAAgaEMAcgB5AHAAdABvAGcAcgBhAHAAaABpA ZAB1AHIDgYkAUL9RtrwlnPb5Ys6tk8N+R/ABN/KEC++h4 vLGCmQrAdxNc/gcC5nvDUoaciL1I4dIAcZh/ew4jhNqM7: XIqGaJihWa5KciQvIwF9Nhtw3ciilyQQR8M+iseGxgc+S ADANBgkqhkiG9w0BAQUFAAOBgQCuB8x1rscQLGVyQGpym 5TpUmFyC/NKrQMI7DaKteNphi7FzD1Cuf2tNHYuikt85 uPVc/1IEvvDgbjy7JFYf102jK8VtWb7Bd5MJzvpLJK56 sIGUzw== -----END NEW CERTIFICATE REQUEST-----</pre>



Click the CONTINUE button to submit the CSR and proceed with the Test Server ID enrollment.

Continue

Figure 9-21. Submitting your CSR

You're now on step 3 of 5 through which you get to review and complete your application with VeriSign. Double-check the data in the Distinguished Name box, enter technical contact information in the box below and then click the Accept button. You will then be presented with a Web page telling you that VeriSign is now processing your request and will be sending you an e-mail in the next hour that

contains your trial Server ID and instructions for installation. This is probably a good time to go take a break; I know I would.

Okay, the hour's up and your e-mail from VeriSign should have arrived by now. At the bottom of this e-mail is your certificate response from VeriSign. To get things going, create a new text file on your hard drive called `certres.txt`. Then copy and paste the certificate response found at the bottom of your e-mail. Make sure to include `BEGIN CERTIFICATE` and `END CERTIFICATE` in the text file. Once you've closed and saved this certificate response text file, reopen the Internet Services Manager, right-click the Default Web Site, and select Properties as you've done before. Select the Directory Security tab and then click the Server Certificate button. This will bring up the familiar Web Server Certificate Wizard in which you'll need to click Next to proceed. On the Pending Certificate Request screen, shown in Figure 9-22, you'll need to select "Process the pending request and install the certificate" and then click Next.

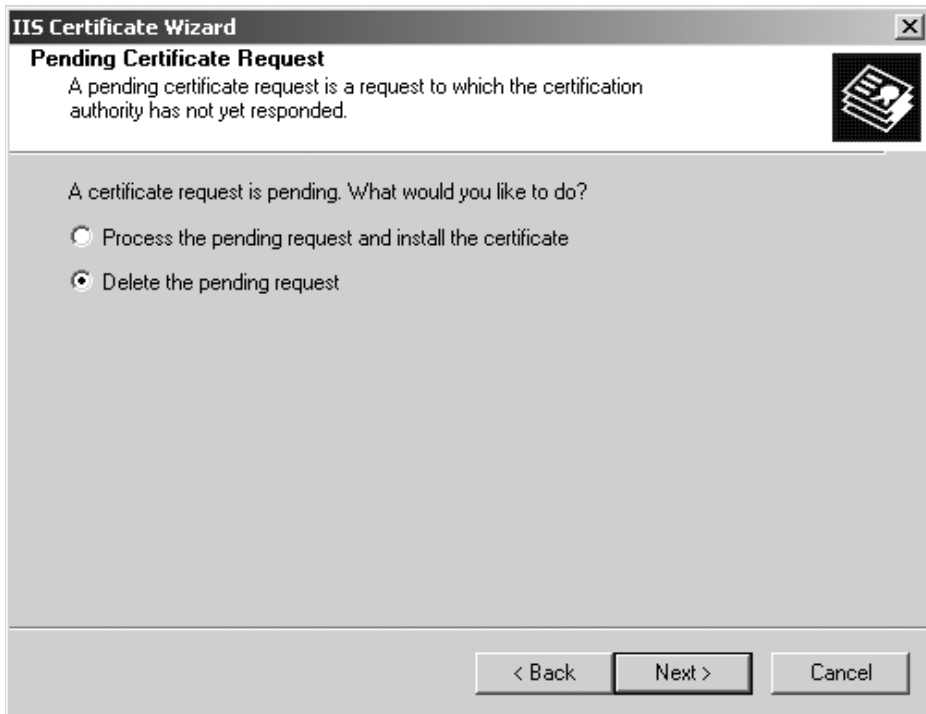


Figure 9-22. Pending certificate request

On the Process a Pending Request screen shown in Figure 9-23, click the Browse button to navigate to the certres.txt file you just created and click Next on this and the following screen.

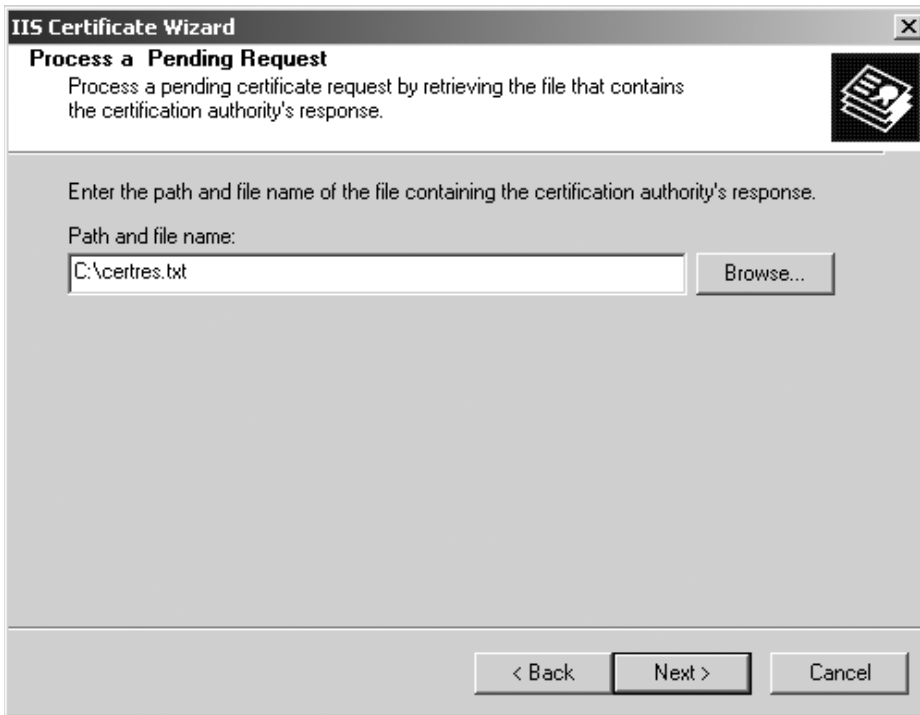


Figure 9-23. Processing a pending request

Your certificate is now installed on IIS. The next step is to enable SSL for your FieldAgentRDA Web site. To do this, right-click FieldAgentRDA while still inside Internet Information Services, and select Properties. Select the Directory Security tab and click the Edit button in the Secure Communications area. In the Secure Communications dialog box shown in Figure 9-24, check the Require secure channel (SSL) checkbox, and then click OK.

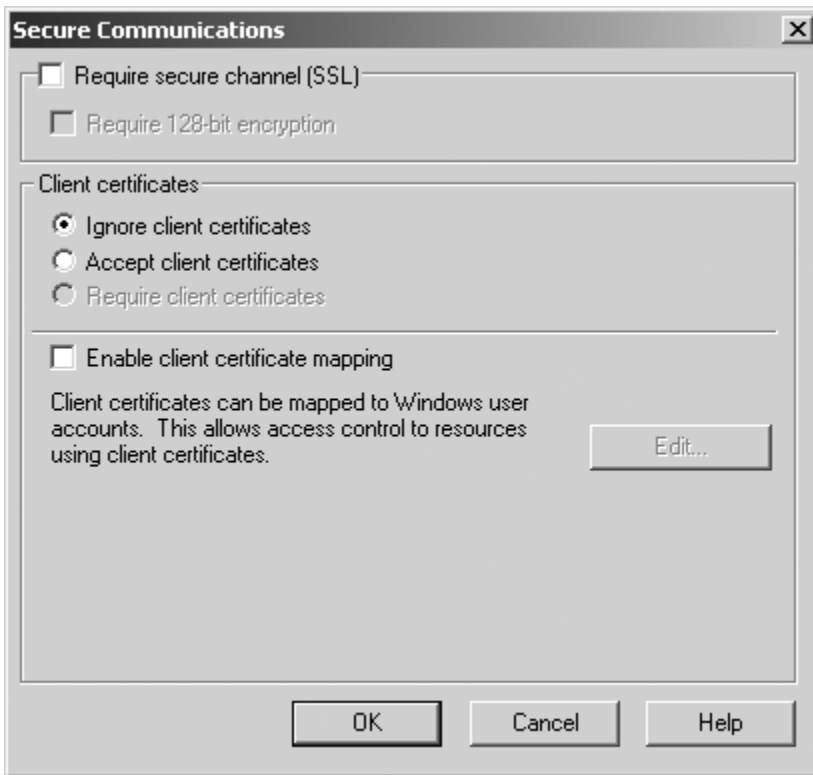


Figure 9-24. Securing communications

By now you've probably lost all patience and you're dying to see if you can hit your FieldAgentRDA Web site via Pocket Internet Explorer. Keep in mind that VeriSign made it abundantly clear that you'll need to download a Test CA Root in order for you to view the site without receiving some kind of an invalid certificate error. Just for fun, enter `https://servername/fieldagentrda/sscesa20.dll` in Pocket Internet Explorer to view this certificate error as shown in Figure 9-25.



Figure 9-25. Security error

If you click Yes in this Security dialog box, you should be able to enter your Basic Authentication username and password and view the site securely. Obviously, this isn't good enough. In order to have things operate smoothly, you will need to download and install the VeriSign Test CA Root. Navigate your desktop browser to <http://www.verisign.com/server/trial/faq/index.html>, click the Accept button, and download the file called `getcacert.cer` to your hard drive. Additionally, you need to download a Pocket PC program call `AddRootCert` from Microsoft at <http://download.microsoft.com/download/pocketpc/addroot/1.0/wce/en-us/AddRootCert.exe>. This program is used to add additional certificates to your Pocket PC. It comes as a self-extracting WinZip EXE that you'll need to unzip into a temporary directory on your desktop. Once you've unzipped it, look inside a folder called `Armrel` in order to find the actual `AddRootCert.exe` program that you're looking for. Once that's been accomplished, copy both `getcacert.cer` and `AddRootCert.exe` to the `/My Documents` directory of your Pocket PC via the Explore feature of ActiveSync. Once those files are copied over, open the File Explorer on your Pocket PC and tap `AddRootCert.exe`. Once the application is running, click the File menu, and select Open in order to search for your VeriSign certificate. Once found, click `getcacert` and the `AddRootCert` application will look like Figure 9-26.



Figure 9-26. AddRootCert Powertoy

Click the Install Certificate button to add the certificate to your device. Now comes the moment of truth. After making sure that Pocket Internet Explorer has been removed from memory, relaunch it and navigate to the FieldAgentRDA Web site. If your installation has gone well, you won't see an invalid certificate warning on your device and you'll be good to go for secure communications via both RDA and merge replication.

Creating the Field Memo Database

One of the great things about RDA, and merge replication for that matter, is that you can design your SQL Server CE database on the desktop or server using SQL Server's Enterprise Manager. I'm sure you'll find this much easier than using either Query Analyzer or DDL on the handheld. All aspects of the database structure you create are faithfully downloaded and reproduced on your Pocket PC. The first thing you need to do to get started with this RDA project is to build the database

that will contain the memos sent in from the field. Even though you can use SQL Server 6.5 or 7.0 for this task, I'd prefer you use SQL Server 2000, since you'll use the same database in Chapter 10's examples. On that note, fire up Enterprise Manager and create a database called IntelligenceData.

A Table, Columns, and Indexes

I'm going to make things real easy for you when it comes to creating this database application. For starters, there's only one table with nine columns, so there's not much to think about. In the Enterprise Manager, right-click your IntelligenceData database and select New Table. This table will be called FieldMemos, and its columns are described in Table 9-4.

Table 9-4. FieldMemos

COLUMNNAME	DATATYPE	LENGTH	NULLS	PRIMARY KEY	INDEXED	DEFAULT VALUE
MemoID	uniqueidentifier	16	No	Yes	Yes	newid()
MemoTo	nvarchar	50	Yes	No	Yes	
MemoFrom	nvarchar	50	Yes	No	Yes	
MemoDateTime	datetime	8	Yes	No	Yes	
MemoCity	nvarchar	50	Yes	No	Yes	
MemoState	nvarchar	2	Yes	No	Yes	
MemoSubject	nvarchar	50	Yes	No	Yes	
MemoPriority	int	4	Yes	No	Yes	
MemoBody	ntext	16	Yes	No	Yes	

Figure 9-27 gives you the design view of the FieldMemos table inside Enterprise Manager.

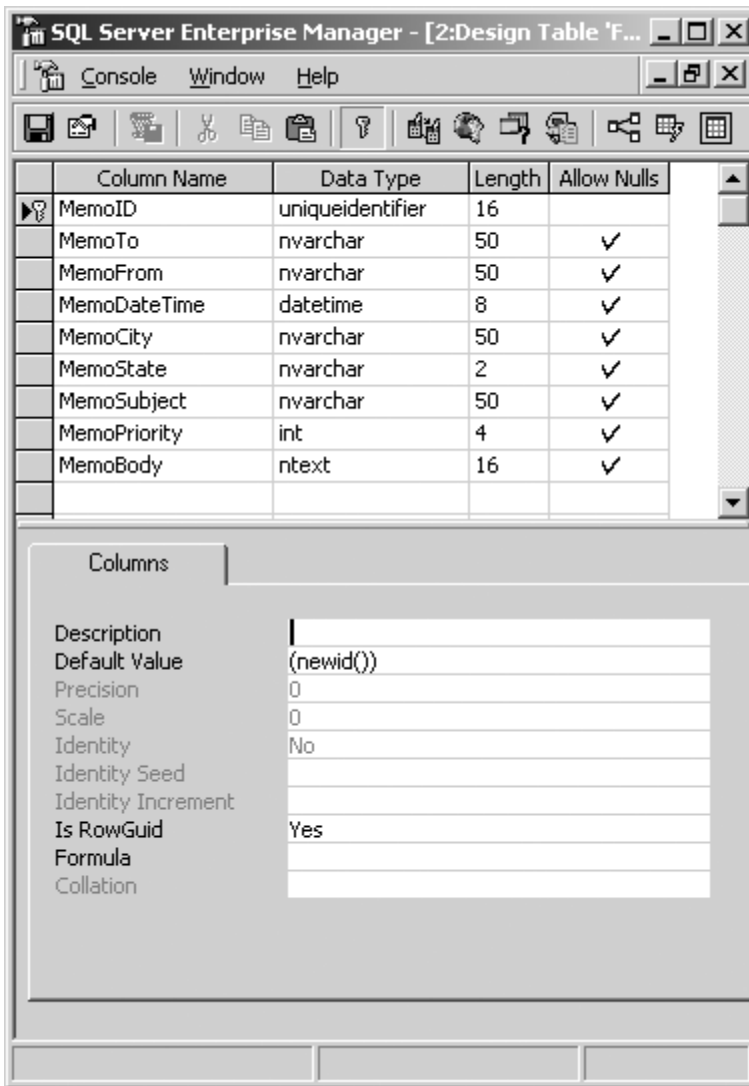


Figure 9-27. Field memos in design mode

Primary Key Concerns

The first strange thing you probably noticed is that my primary key column, MemoID, is a uniqueidentifier. You're probably accustomed to using the integer data type along with an IDENTITY constraint in order to have an auto-incrementing primary key column. In a situation where multiple clients are all making inserts to the same table, this is perfectly acceptable. Just think about the RDA architecture for a moment. Multiple users are pulling down copies of the same table for offline

use in the field. RDA is using optimistic concurrency so none of the pulled records are locked on the server. Offline users are making numerous inserts of new data to their offline tables. If IDENTITY columns are being employed for the primary keys of those offline tables, you'd have lots of MemoIDs incrementing along to their heart's content. With each new memo added to the offline database, the primary key is incremented one, two, three, four, five, etc. Can you imagine thousands of users all sharing the same values for their MemoID column? Everything is cool until these users attempt to push their new rows of data back to the primary SQL Server database. SQL exceptions will be flying when all those identical primary keys collide as SQL Server tries to merge these field memos into the database. Therefore, this method of trying to keep all your new rows unique is flawed, because you can't guarantee their uniqueness across multiple offline SQL Server CE databases. The only way to guarantee complete uniqueness in any situation is to employ globally unique identifiers as your primary key data type. When coupled with the `newid()` function as your default value to generate new GUIDs each time a row is added to the table, your row's uniqueness is ensured.

Table Structure Explanation

A quick glance at the structure of this table reveals that it looks like the fields found in an e-mail. The MemoTo field is usually filled with "Office of the Director", whereas the MemoFrom field is filled in with the name of the agent in the field. The MemoDateTime field will contain a system-generated timestamp; and the MemoCity and MemoState fields will reveal where the memo is coming from. The MemoSubject field is pretty self-explanatory and the MemoPriority field will contain 1, 2, or 3 to denote the priority level. In the MemoBody field, I chose to use the `ntext` data type since these field memos can be quite detailed and lengthy, and I didn't want to limit the length of the content to 255 characters by using the `nvarchar` data type.

Creating the RDA Client

After all this IIS, security, and SQL Server configuration, you finally get to write some code and work with the RDA object model. Luckily, all this up-front work you just completed is utilized by merge replication so you won't have to do this all over again in Chapter 10. Earlier in this chapter, I gave you a quick overview of the Pull, Push, and SubmitSQL methods of the `SqlCeRemoteDataAccess` class. In fact, they're the only three methods you'll find in the class, so there's no danger of things getting too complicated. This class has a number of properties as described in

Table 9-5 whose values are essential to the smooth execution of the three different methods.

Table 9-5. SqlCeRemoteDataAccess Properties

PROPERTY	DESCRIPTION
InternetLogin	IIS login name used when connecting to the SQL Server CE Server Agent
InternetPassword	IIS password used when connecting to the SQL Server CE Server Agent
InternetProxyLogin	Login name used when connecting to a proxy server that requires authentication
InternetProxyPassword	Password used when connecting to a proxy server that requires authentication
InternetProxyServer	The proxy server to use when accessing the HTTP resource specified in the InternetUrl property
InternetUrl	The URL used to connect to the SQL Server CE Server Agent
LocalConnectionString	The OLE DB connection string for the SQL Server CE database residing on the handheld

InternetLogin and InternetPassword correlate to the Basic Authentication credentials you have to send to IIS in order to access the SQL Server CE Server Agent. If you're using a proxy server in your network, then you'll specify it in the InternetProxyServer property. If that proxy server requires authentication, you'll take care of that using the InternetProxyLogin and InternetProxyPassword properties. Remember how I had you point to the sscesa20.dll file in the FieldAgentRDA virtual directory? Well, the InternetUrl property is where you assign that URL. I find the LocalConnectionString property a little strange in that it wants you to use an OLE DB connection string to communicate with your SQL Server CE database rather than using ADO.NET. I'm sure that will change in future versions of the RDA technology.

Time to Code

Bring up Visual Studio .NET 2003 and create a Smart Device Application in either C# or Visual Basic .NET. In the spirit of keeping things simple, I want you to add just four buttons to your form as described in Table 9-6.

Table 9-6. Buttons

NAME	TEXT
btnPull	Pull
btnAddMemo	Add Memo
btnPush	Push
btnSubmitSQL	Submit SQL

You can place these buttons anywhere on the form that you like. The purpose of these buttons is to illustrate how to pull data down to your device, add new rows of data to SQL Server CE, push the data back up to the server, and submit queries for execution on SQL Server. The code to make all this possible will be placed in the click events of these four buttons. Before you write any code, make sure to add a reference to both `System.Data.Common` and `System.Data.SqlServerCe`. At the top of your form class, add

```
using System.Data.SqlServerCe;
```

and

```
using System.IO;
```

if you're doing a C# project.

If you're creating a Visual Basic .NET project, add

```
Imports System.Data.SqlServerCe
```

and

```
Imports System.IO
```

Pull

You already know that the Pull method will retrieve tables, indexes, and data from SQL Server and transfer it to SQL Server CE on your Pocket PC. What you don't know is that the Pull method has several important arguments that are critical to making the retrieval go off without a hitch, and these are shown in Table 9-7.

Table 9-7. Pull Method Arguments

ARGUMENT	DESCRIPTION
LocalTableName	Name of the SQL Server CE table that will be created and populated with data as a result of the Pull operation
SqlSelectString	Query that determines which rows and columns are retrieved from SQL Server
OLEDBConnectionString	An OLE DB–based connection string needed to connect to the proper remote SQL Server database
TrackOption	Indicates to the SQL Server CE Database Engine whether or not to track changes made to the local tables and whether or not to re-create the indexes that the tables contain in SQL Server
ErrorTable	Name of the SQL Server CE table used for error purposes

Usually, LocalTableName will be the same name as the name of the table you're pulling from on SQL Server. If you wish to retrieve all the data from a given SQL Server table, then you'll use a SELECT * statement. OLEDBConnectionString will take you back in time to the days of classic ADO. You'll specify SQLOLEDB as the provider, the data source will be the name of the server that SQL Server is running on, the initial catalog will be the name of the actual database, and the user ID and password are the required credentials based on SQL Server Authentication.

There are potentially two other things you need to concern yourself with before setting the RDA properties and calling the Pull method. It's usually a good idea to check whether your SQL Server CE database even exists before attempting this operation. The reason is that an RDA Pull method is usually relied upon to populate Pocket PC devices with databases and tables to help streamline deployment. Even though the Pull method only creates tables, it's a good idea to check for the existence of a database and then create one if it isn't there. If, on the other hand, the database already exists, it's possible that the table that the Pull method is about to create is already present in the database. Since the Pull method is somewhat destructive in that it isn't designed to merge server data with your handheld, you need to drop the local table before running the Pull method, or an error will occur. The following example that resides in the click event of the Pull

button illustrates these issues and shows you how to properly run the Pull method in order to create the FieldMemos table in your local database.

Visual Basic .NET

```
Private Sub btnPull_Click(ByVal sender As System.Object, & _
                          ByVal e As System.EventArgs) Handles btnPull.Click

    Dim cn As SqlConnection
    Dim rda As SqlCeRemoteDataAccess
    Dim sqlEngine As SqlCeEngine

    Try
        btnPull.Enabled = False

        ' Create database if it doesn't already exist
        If (Not File.Exists("My Documents\IntelligenceData.sdf")) Then
            sqlEngine = New SqlCeEngine
            sqlEngine.LocalConnectionString = & _
                "Data Source=My Documents\IntelligenceData.sdf;" & _
                "Password=apress;" & _
                "Encrypt Database=True"
            sqlEngine.CreateDatabase()
            sqlEngine.Dispose()
        Else
            ' Open the connection to the database
            cn = New SqlConnection("Data Source=My Documents" & _
                                   "\IntelligenceData.sdf;Password=apress")
            cn.Open()
            Dim cmd As SqlCommand = cn.CreateCommand()

            ' Drop the FieldMemos table
            cmd.CommandText = "DROP TABLE FieldMemos"
            cmd.ExecuteNonQuery()

            ' Close the connection
            If cn.State <> ConnectionState.Closed Then
                cn.Close()
            End If
        End If

        ' Instantiate the RDA Object
        rda = New SqlCeRemoteDataAccess
```

```

' Connection String to the SQL Server.
Dim remoteConnectionString As String = "Provider=SQLOLEDB;" & _
    "Data Source=seawolf;" & _
    "Initial Catalog=IntelligenceData;" & _
    "User Id=sa;" & _
    "Password=apress"

rda.InternetLogin = "robtiffany"
rda.InternetPassword = "pinnacle"
rda.InternetUrl = "https://seawolf/fieldagentrda/sscesa20.dll"
rda.LocalConnectionString = "Data Source=\My Documents\" & _
    "IntelligenceData.sdf;" & _
    "SSCE:Database Password=apress"

rda.Pull("FieldMemos", "Select * from FieldMemos", remoteConnectionString, & _
    RdaTrackOption.TrackingOnWithIndexes, "FieldMemosErrorTable")

Catch sqllex As SqlCeException
    Dim sqlError As SqlCeError
    For Each sqlError In sqllex.Errors
        MessageBox.Show(sqlError.Message)
    Next
Catch ex As Exception
    MessageBox.Show(ex.Message)
Finally
    rda.Dispose()
    btnPull.Enabled = True
End Try
End Sub

```

C#

```

private void btnPull_Click(object sender, System.EventArgs e)
{
    SqlCeConnection cn = null;
    SqlCeRemoteDataAccess rda = null;
    SqlCeEngine sqlEngine = null;

    try
    {
        btnPull.Enabled = false;
    }
}

```



```

// Create database if it doesn't already exist
if (!File.Exists(@"\My Documents\IntelligenceData.sdf"))
{
    sqlEngine = new SqlCeEngine();
    sqlEngine.LocalConnectionString = "Data Source=\\My Documents\\" +
        "IntelligenceData.sdf;" +
        "Password=apress;" +
        "Encrypt Database=True";

    sqlEngine.CreateDatabase();
    sqlEngine.Dispose();
}
else
{
    // Open the connection to the database
    cn = new SqlCeConnection("Data Source=\\My Documents\\" +
        "IntelligenceData.sdf;Password=apress");

    cn.Open();
    SqlCeCommand cmd = cn.CreateCommand();

    // Drop the FieldMemos table
    cmd.CommandText = "DROP TABLE FieldMemos";
    cmd.ExecuteNonQuery();

    // Close the connection
    if (cn.State != ConnectionState.Closed)
    {
        cn.Close();
    }
}

// Instantiate the RDA Object
rda = new SqlCeRemoteDataAccess();

// Connection String to the SQL Server.
string remoteConnectionString = "Provider=SQLOLEDB;" +
    "Data Source=seawolf;" +
    "Initial Catalog=IntelligenceData;" +
    "User Id=sa;" +
    "Password=apress";

rda.InternetLogin = "robtiffany";
rda.InternetPassword = "pinnacle";
rda.InternetUrl = "https://seawolf/fieldagentrda/sscesa20.dll";
rda.LocalConnectionString = "Data Source=\\My Documents\\" +

```

```

        "IntelligenceData.sdf;" +
        "SSCE:Database Password=apress";

        rda.Pull("FieldMemos",
            "Select * from FieldMemos",
            remoteConnectString,
            RdaTrackOption.TrackingOnWithIndexes,
            "FieldMemosErrorTable");
    }
    catch(SqlCeException sqlcx)
    {
        foreach(SqlCeError sqlError in sqlcx.Errors)
        {
            MessageBox.Show(sqlError.Message, "Error");
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message, "Error");
    }
    finally
    {
        rda.Dispose();
        btnPull.Enabled = true;
    }
}

```

Add Memo

Once you have a database, table, and indexes created on your Pocket PC, you need to start adding your field memos to it. The following example resides inside the click event of the AddMemo button and shows you how to insert your memo data into the database. You could even build a full-blown graphical application with text boxes tied to the parameters of the SqlCeCommand object to drive this insert statement rather than use the static information I've included.

Visual Basic .NET

```

Private Sub btnAddMemo_Click(ByVal sender As System.Object, & _
    ByVal e As System.EventArgs) Handles btnAddMemo.Click

    Dim cn As SqlCeConnection

```

```

Try
    btnAddMemo.Enabled = False
    cn = New SqlCeConnection("Data Source=\My
Documents\IntelligenceData.sdf;Password=apress")
    cn.Open()

    Dim cmd As SqlCeCommand = cn.CreateCommand()

    ' Insert Memo data
    cmd.CommandText = "INSERT INTO FieldMemos (" & _
        "MemoTo, " & _
        "MemoFrom, " & _
        "MemoDateTime, " & _
        "MemoCity, " & _
        "MemoState, " & _
        "MemoSubject, " & _
        "MemoPriority, " & _
        "MemoBody) " & _
        "VALUES " & _
        "(?, ?, ?, ?, ?, ?, ?, ?)"
    cmd.Parameters.Add("@MemoTo", "Office of the Director")
    cmd.Parameters.Add("@MemoFrom", "Agent 99")
    cmd.Parameters.Add("@MemoDateTime", System.DateTime.Now)
    cmd.Parameters.Add("@MemoCity", "Houston")
    cmd.Parameters.Add("@MemoState", "TX")
    cmd.Parameters.Add("@MemoSubject", "Suspicious Activity 2")
    cmd.Parameters.Add("@MemoPriority", 1)
    cmd.Parameters.Add("@MemoBody", "Observed a suspicious meeting")
    cmd.Prepare()
    cmd.ExecuteNonQuery()

Catch sqlEx As SqlCeException
    Dim sqlError As SqlCeError
    For Each sqlError In sqlEx.Errors
        MessageBox.Show(sqlError.Message)
    Next
Catch ex As Exception
    MessageBox.Show(ex.Message)
Finally
    If cn.State <> ConnectionState.Closed Then
        cn.Close()
    End If
    btnAddMemo.Enabled = True
End Try
End Sub

```

C#

```

private void btnAddMemo_Click(object sender, System.EventArgs e)
{
    SqlConnection cn = null;

    try
    {
        btnAddMemo.Enabled = false;
        cn = new SqlConnection("Data Source=\\My Documents\\" +
                               "IntelligenceData.sdf;Password=apress");

        cn.Open();

        SqlCommand cmd = cn.CreateCommand();

        // Insert Memo data
        cmd.CommandText = "INSERT INTO FieldMemos (" +
                           "MemoTo, " +
                           "MemoFrom, " +
                           "MemoDateTime, " +
                           "MemoCity, " +
                           "MemoState, " +
                           "MemoSubject, " +
                           "MemoPriority, " +
                           "MemoBody) " +
                           "VALUES " +
                           "(?, ?, ?, ?, ?, ?, ?, ?)";

        cmd.Parameters.Add("@MemoTo", "Office of the Director");
        cmd.Parameters.Add("@MemoFrom", "Agent 86");
        cmd.Parameters.Add("@MemoDateTime", System.DateTime.Now);
        cmd.Parameters.Add("@MemoCity", "Houston");
        cmd.Parameters.Add("@MemoState", "TX");
        cmd.Parameters.Add("@MemoSubject", "Suspicious Activity");
        cmd.Parameters.Add("@MemoPriority", 1);
        cmd.Parameters.Add("@MemoBody", "Observed a suspicious meeting");
        cmd.Prepare();
        cmd.ExecuteNonQuery();
    }
    catch (SqlCeException sqllex)
    {
        foreach(SqlCeError sqlError in sqllex.Errors)
        {
            MessageBox.Show(sqlError.Message, "Error");
        }
    }
}

```

```

    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message, "Error");
    }
    finally
    {
        if(cn.State != ConnectionState.Closed)
        {
            cn.Close();
        }
        btnAddMemo.Enabled = true;
    }
}

```

Push

Calling the Push method doesn't require nearly as much coding as the Pull method. There are no databases to create or tables to drop in this case. The Push method only has three arguments that you need to concern yourself with, as described in Table 9-8.

Table 9-8. Push Method Arguments

ARGUMENT	DESCRIPTION
LocalTableName	Name of the previously pulled and tracked table whose data needs to be sent back to SQL Server for updates.
OleDbConnectionString	An OLE DB-based connection string needed to connect to the proper remote SQL Server database.
BatchOption	You specify <code>BatchingOn</code> or <code>BatchingOff</code> to control whether to treat all the rows of data being sent to SQL Server as a single transaction or to process each row individually

The following example that resides inside the click event of the Push button shows you how to send the new field memos that you generated back to SQL Server for analysis. I use the `BatchingOn` option in the Push method because I want all the rows of data in the Push operation to either succeed or fail as a single transaction.

Visual Basic .NET

```

Private Sub btnPush_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnPush.Click

    Dim rda As SqlCeRemoteDataAccess

    Try

        btnPush.Enabled = False

        'Instantiate the RDA Object
        rda = New SqlCeRemoteDataAccess

        'Connection String to the SQL Server.
        Dim remoteConnectionString As String = "Provider=SQLOLEDB;" & _
        "Data Source=seawolf;" & _
        "Initial Catalog=IntelligenceData;" & _
        "User Id=sa;" & _
        "Password=apress"

        rda.InternetLogin = "robtiffany"
        rda.InternetPassword = "pinnacle"
        rda.InternetUrl = "https://seawolf/fieldagentrda/sscesa20.dll"
        rda.LocalConnectionString = & _
        "Data Source=\My Documents\IntelligenceData.sdf;" & _
        "SCE:Database Password=apress"

        rda.Push("FieldMemos", remoteConnectionString, RdaBatchOption.BatchingOn)

    Catch sqllex As SqlCeException
        Dim sqlError As SqlCeError
        For Each sqlError In sqllex.Errors
            MessageBox.Show(sqlError.Message)
        Next
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Finally
        rda.Dispose()
        btnPush.Enabled = True
    End Try
End Sub

```

C#

```

private void btnPush_Click(object sender, System.EventArgs e)
{
    SqlCeRemoteDataAccess rda = null;

    try
    {
        btnPush.Enabled = false;

        // Instantiate the RDA Object
        rda = new SqlCeRemoteDataAccess();

        // Connection String to the SQL Server.
        string remoteConnectionString = "Provider=SQLOLEDB;" +
                                         "Data Source=seawolf;" +
                                         "Initial Catalog=IntelligenceData;" +
                                         "User Id=sa;" +
                                         "Password=apress";

        rda.InternetLogin = "robtiffany";
        rda.InternetPassword = "pinnacle";
        rda.InternetUrl = "https://seawolf/fieldagentrda/sscesa20.dll";
        rda.LocalConnectionString = "Data Source=\\My Documents\\" +
                                     "IntelligenceData.sdf;" +
                                     "SSCE:Database Password=apress";

        rda.Push("FieldMemos", remoteConnectionString, RdaBatchOption.BatchingOn);
    }
    catch(SqlCeException sqllex)
    {
        foreach(SqlCeError sqlError in sqllex.Errors)
        {
            MessageBox.Show(sqlError.Message, "Error");
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message, "Error");
    }
    finally

```

```

    {
        rda.Dispose();
        btnPush.Enabled = true;
    }
}

```

Submit SQL

The SubmitSQL method doesn't have much to do with synchronizing data between a handheld and a server. I personally think it was just thrown in there for good measure since the architecture already exists to support the remote execution of SQL statements against SQL Server. You're allowed to execute any kind of query or stored procedure, as long as it doesn't involve the returning of rows back to the Pocket PC. Table 9-9 explains the usage of the two arguments for the SubmitSQL method.

Table 9-9. SubmitSQL Method Arguments

ARGUMENT	DESCRIPTION
SQLString	Any valid SQL statement that doesn't return rows
OleDbConnection String	An OLE DB-based connection string needed to connect to the proper remote SQL Server database

In the following example that resides inside the click event of the Submit SQL button, I use the same insert statement that I showed you back in the Add Memo example to submit the field memo directly into SQL Server without first saving it to the local SQL Server CE database. Keep in mind that this method only works in a reliable, network-connected state. It's definitely not as robust a solution as the offline design of Pull and Push.

Visual Basic .NET

```

Private Sub btnSubmitSQL_Click(ByVal sender As System.Object, & _
    ByVal e As System.EventArgs) Handles submitSQL.Click

    Dim rda As SqlCeRemoteDataAccess

    Try
        btnSubmitSQL.Enabled = False
    
```



```

' Instantiate the RDA Object
rda = New SqlCeRemoteDataAccess

' Connection String to the SQL Server.
Dim remoteConnectionString As String = "Provider=SQLOLEDB;" & _
    "Data Source=seawolf;" & _
    "Initial Catalog=IntelligenceData;" & _
    "User Id=sa;" & _
    "Password=apress"

rda.InternetLogin = "robtiffany"
rda.InternetPassword = "pinnacle"
rda.InternetUrl = "https://seawolf/fieldagentrda/sscesa20.dll"
rda.LocalConnectionString = "Data Source=\My Documents\" & _
    "IntelligenceData.sdf;" & _
    "SSCE:Database Password=apress"

Dim remoteSQL As String = "INSERT INTO FieldMemos (" & _
    "MemoTo, " & _
    "MemoFrom, " & _
    "MemoDateTime, " & _
    "MemoCity, " & _
    "MemoState, " & _
    "MemoSubject, " & _
    "MemoPriority, " & _
    "MemoBody) " & _
    "VALUES " & _
    "('Office of the Director', " & _
    "'Agent 99', " & _
    "'" + System.DateTime.Now + "', " & _
    "'Houston', " & _
    "'TX', " & _
    "'Suspicious Activity 3', " & _
    "'1', " & _
    "'Observed a suspicious meeting')"
```

```

rda.SubmitSql(remoteSQL, remoteConnectionString)

Catch sqllex As SqlCeException
    Dim sqlError As SqlCeError
    For Each sqlError In sqllex.Errors
        MessageBox.Show(sqlError.Message)
    Next
Catch ex As Exception

```

```

        MessageBox.Show(ex.Message)
    Finally
        rda.Dispose()
        btnSubmitSQL.Enabled = True
    End Try
End Sub

```

C#

```

private void btnSubmitSQL_Click(object sender, System.EventArgs e)
{
    SqlCeRemoteDataAccess rda = null;

    try
    {
        btnSubmitSQL.Enabled = false;

        // Instantiate the RDA Object
        rda = new SqlCeRemoteDataAccess();

        // Connection String to the SQL Server.
        string remoteConnectionString = "Provider=SQLOLEDB;" +
            "Data Source=seawolf;" +
            "Initial Catalog=IntelligenceData;" +
            "User Id=sa;" +
            "Password=apress";

        rda.InternetLogin = "robtiffany";
        rda.InternetPassword = "pinnacle";
        rda.InternetUrl = "https://seawolf/fieldagentrda/sscesa20.dll";
        rda.LocalConnectionString = "Data Source=\\My Documents\\" +
            "IntelligenceData.sdf;" +
            "SSCE:Database Password=apress";

        string remoteSQL = "INSERT INTO FieldMemos (" +
            "MemoTo, " +
            "MemoFrom, " +
            "MemoDateTime, " +
            "MemoCity, " +
            "MemoState, " +
            "MemoSubject, " +
            "MemoPriority, " +
            "MemoBody) " +

```

```

        "VALUES " +
        "('Office of the Director', " +
        "'Agent 99', " +
        "'" + System.DateTime.Now + "', " +
        "'Houston', " +
        "'TX', " +
        "'More Suspicious Activity', " +
        "'1', " +
        "'Observed another suspicious meeting');"

        rda.SubmitSql(remoteSQL, remoteConnectionString);
    }
    catch (SqlCeException sqllex)
    {
        foreach (SqlCeError sqlError in sqllex.Errors)
        {
            MessageBox.Show(sqlError.Message, "Error");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error");
    }
    finally
    {
        rda.Dispose();
        btnSubmitSQL.Enabled = true;
    }
}

```

Run the Pull, AddMemo, and Push operations several times in succession while changing the memo data each time. Take a look at the changes in data rows on the server with Enterprise Manager and on the Pocket PC with Query Analyzer. I think you'll find that this is a very effective method of working with, adding to, and updating offline data and synchronizing it with SQL Server.

Conclusion

It probably seems like setting up and executing an RDA solution is a lot of work for an object that only has three methods. I hope that by taking you step by step through each configuration process at a granular level, you'll come away with a better understanding of how RDA works. It was also very important for you to learn the process of securing your RDA system with SSL, since this step is usually

overlooked but extremely important. I'm sure you're glad to see that you can design your SQL Server CE databases at the SQL Server level inside Enterprise Manager and then download the results to your handheld. That sure beats using DDL or Query Analyzer on your Pocket PC to do the same thing. The next and final chapter covers the synchronization of your field memo table using merge replication. Luckily, all the work you've done in this chapter to create HTTPS connectivity between your Pocket PC and IIS will be reused with merge replication. With a little more configuration required on SQL Server but significantly less code on the client, I'm sure you'll find this technology to be very interesting.