7

Configuring SQL Server Network Communication

SQL Server 2008 is a client-server application designed to efficiently exchange data and instructions over multiple network connections. Understanding the network connections and how they can be configured is a big part of a DBA's job. Microsoft has made your job easier by minimizing the number of network protocols that SQL Server 2008 supports to the most commonly implemented protocols, but at the same time, the job of the DBA is made more complex by the ability to configure multiple connection types with each protocol with the endpoint server object. This chapter discusses the different endpoints that can be configured, as well as the protocol configurations that the endpoints rely on. The chapter also takes a brief look at the client configurations that can be configured with SQL Server 2008.

SQL Server 2008 Network Protocols

SQL Server 2008 provides support for four protocols:

- Shared Memory
- TCP/IP
- Named Pipes
- Virtual Interface Adapter (VIA)

By default, the only network protocols enabled for most editions of SQL Server are TCP/IP and Shared Memory. The Developer and Enterprise Evaluation editions are configured with all protocols except Shared Memory disabled during installation, but the remaining protocols can be enabled if required. If a protocol is not enabled, SQL Server will not listen on an endpoint that is configured to use that protocol. This helps reduce the attack surface of SQL Server.

SQL Server Configuration Manager is used to configure server protocols. With this tool, each supported protocol can be enabled, disabled, and configured as required. The configuration options of the network protocols may not be intuitive, so they justify a little explanation.

Opening SQL Server Configuration Manager displays a node for configuring SQL Server services, SQL Server network protocols, and SQL Native Client protocols. To configure the Server protocols, expand the SQL Server 2008 Network Configuration node and select the instance to be configured. The right-hand pane shows all four of the supported protocols and their status. To display the configurable properties of any of the protocols, double-click on the protocol or right-click on the protocol and select Properties to launch the corresponding Properties window.

Shared Memory

The Shared Memory protocol can only be used by local connections, because it is a shared memory and process space used for inter-server communication. It has only one configurable property: `Enabled`. The `Enabled` property can be set to `Yes` or `No`, resulting in a status of `Enabled` or `Disabled`. Applications or tasks that are designed to run locally on a SQL Server can take advantage of the Shared Memory protocol.

Named Pipes

Named Pipes uses Interprocess Communication (IPC) channels for efficient inter-server communication, as well as local area network (LAN) communication. The Named Pipes protocol has some enhancements in SQL Server 2008 including support for encrypted traffic, but because of the excessive overhead of Named Pipes when connecting across networks or firewalls and the additional port that Named Pipes requires to be opened (445), it is generally a good idea to leave the Named Pipes protocol disabled. However, there are many applications, particularly older applications, that require the Named Pipes protocol because they were designed around NetBIOS or other LAN-based protocols. Named Pipes provides easy access to Remote Procedure Calls (RPC) within a single security domain and thus is advantageous to these applications. If you need to support one of these applications and the SQL Server is not exposed to external traffic, the risk of enabling the Named Pipes protocol and corresponding endpoint is minimal.

Named Pipes has two configurable properties: `Enabled` and `Pipe Name`. The `Enabled` property works the same as the Shared Memory protocol. The `Pipe Name` specifies the inter-process pipe that SQL Server will listen on. The default pipe is `\\.\pipe\MSSQL$<instance_name>\sql\query`.

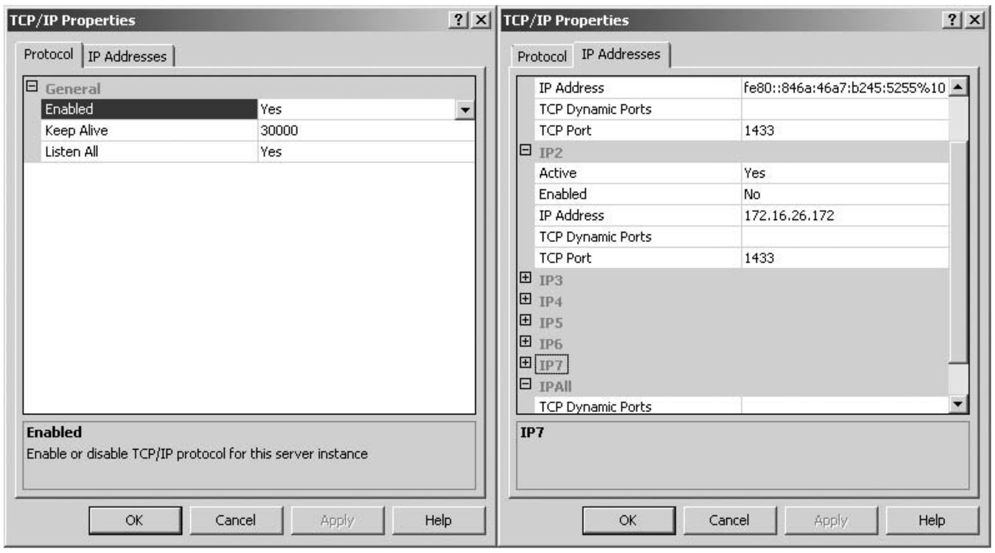
TCP/IP

The TCP/IP protocol is the primary and preferred protocol for most SQL Server installations. It is configured on two separate tabs on the TCP/IP Properties window: the Protocol tab and the IP Addresses tab, as shown in Figure 7-1.

The Protocol tab has the following three configurable properties:

- `Enabled` — This works the same as the other protocols.
- `Keep Alive` — This specifies how many milliseconds SQL Server waits to verify that an idle connection is still valid by sending a `KEEPALIVE` packet. The default is 30,000 milliseconds.
- `Listen All` — This specifies whether SQL Server will listen on all IP addresses configured on the server.

As you can see in Figure 7-1, the "IP Addresses" tab contains configuration settings for each configured IP address on the server and one section for the configuring of all IP addresses. In addition to IPv4, SQL Server 2008 now includes support for IPv6 addresses.


TCP/IP Properties

Protocol | IP Addresses

General	
Enabled	Yes
Keep Alive	30000
Listen All	Yes

Enabled
Enable or disable TCP/IP protocol for this server instance

OK | Cancel | Apply | Help

TCP/IP Properties

Protocol | IP Addresses

IP Address	fe80::846a:46a7:b245:5255%10
TCP Dynamic Ports	
TCP Port	1433
IP2	
Active	Yes
Enabled	No
IP Address	172.16.26.172
TCP Dynamic Ports	
TCP Port	1433
IP3	
IP4	
IP5	
IP6	
IP7	
IPAll	
TCP Dynamic Ports	

IP7

OK | Cancel | Apply | Help

Figure 7-1: Tabs for configuring the TCP/IP protocol.

A detailed explanation of the pros and cons of IPv6 is outside the scope of this book, but you should be aware that it is not uncommon for a single physical adapter to have multiple IPv6 addresses associated with it. Unlike IPv4 addresses, which allow variable-length network masks, IPv6 addresses use fixed-length fields for the network portion and the host portion of the address (each portion uses 64 bits, or half, of the available address). A single host may belong to one or more networks, as defined by the IPv6 protocol. For example, a single host computer will have one IPv6 address to identify it on a non-routed network segment, a second IPv6 address to identify it on a LAN that may include multiple routes, and a third IP address to uniquely identify it on the Internet. What is interesting about IPv6 is that in all cases, the host portion of the address (the second half) stays the same (and is usually a variant of the hardware, or MAC address of the adapter), and the network portion of the address (the first 64 bits) will be different, to identify the networks to which the host machine is connected.

SQL Server will have more than one IPv4 address as well, if only to include the loopback (127.0.0.1) address. All IP addresses, whether version 4 or version 6, are managed from the ''IP Addresses'' tab of the TCP/IP Properties window.

IP address settings are described in the following table:

[redacted]

Continued

[redacted]

Virtual Interface Adapter (VIA)

SQL Server 2008, like its predecessors, also supports the Virtual Interface Adapter protocol, which is used with supported hardware and network configurations. The Virtual Interface Architecture was jointly developed by Compaq (now HP), Intel, and Microsoft, and was designed as a high-performance protocol that could reduce much of the overhead of traditional networking protocols by operating in a user-mode context instead of in a kernel-mode context. VIA network clients connect to a System Area Network (not to be confused with a Storage Area Network, despite the fact that they share an acronym).

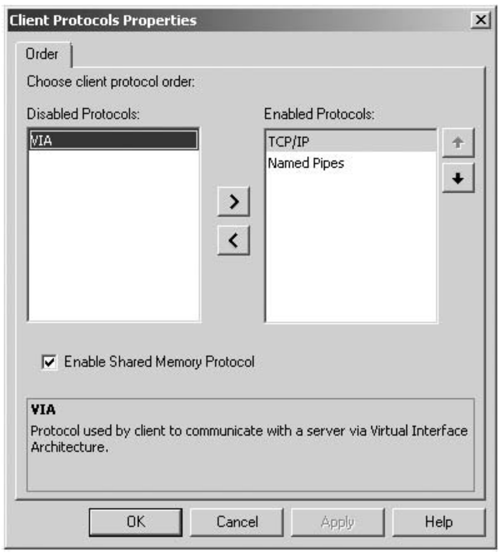
SQL Native Client Configuration

The same four server-side protocols are supported for the SQL Native Client, and, again, SQL Server Configuration Manager is used to enable, disable, or configure these protocols. In addition to the configuration of the client protocols, the binding order of the protocols can also be set. You can do this by expanding the SQL Native Client Configuration node and selecting Client Protocols. In the right-hand pane, right-click on a protocol and select Order to set the order of all enabled protocols, as shown in Figure 7-2.

As Figure 7-2 shows, if the Shared Memory protocol is enabled, it is always first in the binding order. It is not available for manual ordering.

Aliases can be created using the SQL Native Client Configuration. Aliases are very useful in enabling clients to connect to a server even though the name of the server does not match that in the client's connection string. For example, a standby server may be brought up to take the place of a failed server that serves an application with a hard-coded connection string. Without an alias, either the application's connection string would need to be changed, or the server name would have to be changed. By specifying an alias, client requests can be directed to the server without changing the server name. Aliases can also be used to replace a complicated named-instance name.

Figure 7-3 shows the alias YODAHOME being configured for the named instance AUGHTEIGHT\DAGOBAH. To launch the New Alias dialog, right-click on the Aliases node and select ''New Alias.'' Once the alias has been created, new connections can be created by referencing the alias name in lieu of the instance name.



Choose client protocol order:

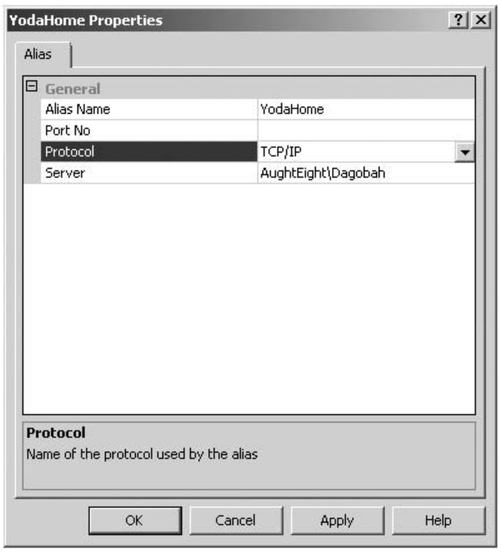
Disabled Protocols:
VIA

Enabled Protocols:
TCP/IP
Named Pipes

☑ Enable Shared Memory Protocol

VIA
Protocol used by client to communicate with a server via Virtual Interface Architecture.

Figure 7-2: Setting the order of enabled protocols.



General	
Alias Name	YodaHome
Port No	
Protocol	TCP/IP
Server	AughtEight\Dagobah

Protocol
Name of the protocol used by the alias

Figure 7-3: Configuring the alias YODAHOME.

SQL Server Endpoints

When SQL Server 2005 came out, there was a lot of confusion and trepidation about SQL Server Endpoints. More to the point, I think a lot of people I worked with were unsure about what they were and why they would use them. The term *endpoint* simply refers to a point of termination on a network, or to be perfectly precise, an *endpoint* is the name for the entity on one end of a transport layer connection. In previous releases of SQL Server, the default network endpoints were UDP port 1434 for the SQL Server Resolution Service and TCP port 1433 for the default instance of SQL Server. Additional TCP ports could be configured for the default and/or any additional named instances. Most database administrators didn't really think of the server listener as an endpoint, but that's what it is, and that's what it will remain. SQL Server 2008 leverages connection objects as endpoints, allowing SQL Server 2008 to listen on different ports, using different transport protocols for different services.

SQL Server provides four different types of endpoints:

- TSQL (both default and TCP)
- Database Mirroring
- SOAP
- Service Broker

Each endpoint provides separate functionality and can be uniquely configured to control access to the Database Engine and associated services.

Default TSQL Endpoints

TSQL endpoints are essentially the same as the standard endpoints that existed in earlier versions of Microsoft SQL Server. During installation, five TSQL endpoints are created:

- TSQL Default TCP
- TSQL Default VIA
- TSQL Named Pipes
- TSQL Local Machine
- Dedicated Administrator Connection (DAC)

The TSQL endpoints are created to provide connection services for the four supported protocols (TCP, VIA, Named Pipes, and Shared Memory). These protocols correspond to the Default TCP, Default VIA, Named Pipes, and Local Machine endpoints. The fifth endpoint created to support the DAC listens on a dedicated TCP port that is configured at start-up to support an administrative connection. The configured port is logged in the current SQL Server log file. (SQL Server log files are described in Chapter 10.)

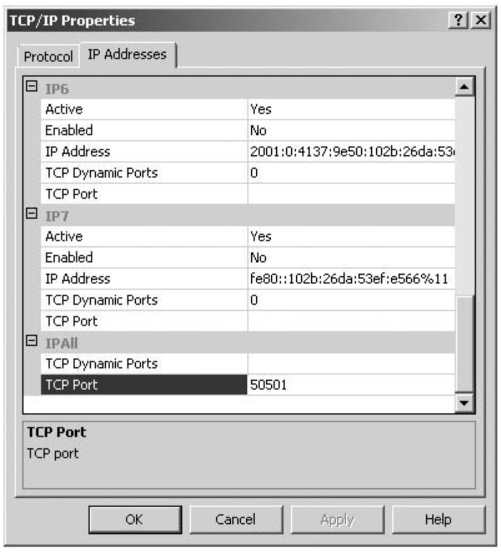
Regardless of the condition of the network protocol, TSQL endpoints have two states: *started* and *stopped*. If the network protocol is enabled and the endpoint is started, SQL Server will listen and accept connections on that endpoint. A stopped endpoint still listens, but actively refuses new connections. If the corresponding protocol is disabled, the TSQL endpoint will not listen and will not respond to client requests.

TSQL endpoints are also known as *Tabular Data Stream* (TDS) endpoints. TDS has been around since Sybase created it in 1984 to support its fledgling relational Database Engine. Microsoft inherited the protocol during its joint venture with Sybase and has since made many changes to the protocol to make it more efficient and secure. It remains the primary protocol for transmitting data from SQL Server to clients via the TCP, Named Pipes, VIA, and Shared Memory protocols.

TSQL Default TCP

The TSQL Default TCP endpoint is created during the installation of a SQL Server instance and is automatically configured to listen on port 1433 for default instances. Named-instance TSQL Default TCP endpoints are randomly assigned a TCP port every time the named instance starts up. However, the port number for named instances can be statically configured with SQL Server Configuration Manager. Configuring a static port can simplify client access and reduce the dependency on the SQL Server Browser Service that enumerates named instances.

To statically configure the port that a named instance of SQL Server will listen on, open SQL Server Configuration Manager, expand the SQL Server 2008 Network Configuration node, and select the instance to configure. Double-click on the TCP/IP protocol in the right-hand pane, or right-click on it and click Properties to launch the TCP/IP Properties window. By default, SQL Server is configured to listen on all available IP addresses, and so the only place that the static port needs to be set is in the IPALL section of the IP Addresses tab on the TCP/IP Properties window (see Figure 7-4). This behavior can be changed by setting the `Listen All` property to `No` on the Protocol tab and individually configuring each IP address.



IP6	
Active	Yes
Enabled	No
IP Address	2001:0:4137:9e50:102b:26da:53
TCP Dynamic Ports	0
TCP Port	
IP7	
Active	Yes
Enabled	No
IP Address	fe80::102b:26da:53ef:e566%11
TCP Dynamic Ports	0
TCP Port	
IPAll	
TCP Dynamic Ports	
TCP Port	50501

TCP Port
TCP port

Figure 7-4: The IPALL section of the IP Addresses tab.

Figure 7-4 shows the TCP port for the named instance DAGOBAH being statically set to port 50101. When configuring ports for named instances, it is best practice to choose a port above 50,000, because many ports below 50,000 are associated with other applications. To retrieve a list of reserved and well-known ports, visit the Internet Assigned Numbers Authority (IANA) web site at `www.iana.org/assignments/port-numbers`.

Keep in mind that the supported protocols are separate from endpoints, and multiple endpoints can be configured for each protocol. In fact, it might be necessary to create multiple TCP endpoints, either for security reasons, such as publishing a SQL Server through a firewall using the non-default port; or for enabling connections that execute against specific processors with Non-Uniform Memory Access (NUMA).

By default, all users have access to the Default TCP endpoint. However, access to the endpoint, as well as any user-created endpoints, can be more tightly controlled with the `GRANT CONNECT` | `DENY CONNECT` | `REVOKE CONNECT` commands.

The state of any endpoint can also be changed with the `ALTER ENDPOINT` command, as shown in the following example:

```
USE Master;
GO
ALTER ENDPOINT [TSQL Default TCP]
STATE=STOPPED;

USE Master;
GO
ALTER ENDPOINT [TSQL Default TCP]
STATE=STARTED;
```

TSQL Default VIA

The VIA protocol is used to support VIA hardware devices such as VIA System Area Networks (SAN). The VIA protocol is dependent on vendor implementations, so a discussion of the VIA endpoint is somewhat difficult without seemingly endorsing one hardware vendor over another. The VIA configurations are usually very straightforward and only require a port assignment. If you are using a VIA hardware implementation for your SAN configuration, make sure you get all the technical documentation you can from your supplier.

TSQL Named Pipes

The Named Pipes endpoint is created to support Named Pipes protocol connections. The Named Pipes protocol was described earlier in this chapter.

TSQL Local Machine

The TSQL Local Machine endpoint allows connections to occur using the Shared Memory protocol. Shared Memory is only accessible on the local machine, hence the TSQL Local Machine designation for this endpoint. Installations of the Enterprise Evaluation and Developer editions of SQL Server 2008 use this endpoint exclusively, unless additional protocols are enabled.

Dedicated Administrator Connection (DAC)

The Dedicated Administrator Connection (DAC) endpoint is used to support limited administrative actions when other connections are unavailable or unresponsive. It uses its own memory area, dedicated TCP port, and CPU scheduler. By default, the DAC endpoint only listens for local connections. Remote DAC connections can be enabled by executing the following code:

```
USE Master;
GO
sp_configure 'remote admin connections', 1;
GO
RECONFIGURE;
GO
```

DAC connections are facilitated through the `SQLCMD` command-line tool.

TSQL TCP Endpoints

In addition to the default TCP endpoints created automatically, additional TSQL TCP endpoints can be created. These TSQL TCP endpoints can be created to support special security or application requirements. However, an important fact to keep in mind is that when a new TSQL TCP endpoint is created, SQL Server automatically revokes all connect permissions to the default endpoint. If connection support is still required for the default endpoint, explicit `GRANT CONNECT` permissions will be necessary to use the default endpoint. SQL Server helps you remember this important fact by always returning a message informing you of the impact of creating a new TCP endpoint, as shown in the next example.

If an additional TSQL TCP endpoint is needed, it can be created using T-SQL. The following example creates an additional TSQL TCP endpoint that is configured to listen on port 50102 and all IP addresses and shows the resulting message warning about permissions:

```
USE Master;
GO
CREATE ENDPOINT DagobahEP
STATE = STARTED
AS TCP
   (LISTENER_PORT = 50102, LISTENER_IP = ALL)
FOR TSQL();
GO

RESULTS:
---------------------------------------------------------------------------
Creation of a TSQL endpoint will result in the revocation of any 'Public' connect
 permissions on the 'TSQL Default TCP' endpoint.  If 'Public' access is desired on
 this endpoint, reapply this permission using 'GRANT CONNECT ON ENDPOINT::[TSQL
 Default TCP] to [public]'.
```

If a single IP address is needed, the `LISTENER_IP` argument can be set to a specific value inside parentheses, as the following example illustrates:

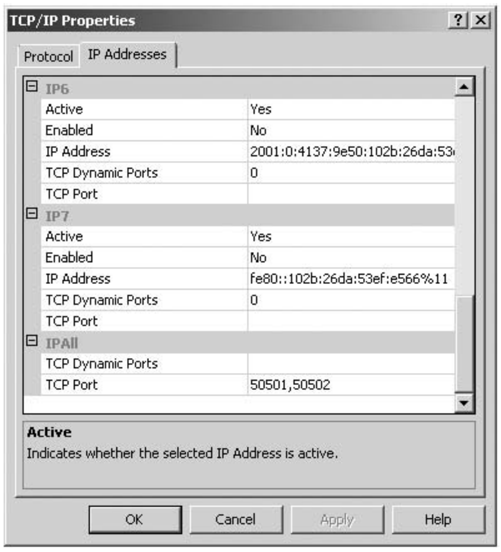
```
USE Master;
GO
CREATE ENDPOINT DagobahEP
STATE = STARTED
AS TCP
   (LISTENER_PORT = 50102, LISTENER_IP = (192.168.1.101))
FOR TSQL();
GO
```

As mentioned earlier, IPv6 is also supported by SQL Server 2008. The address can be configured by passing in the hexadecimal IPv6 address as a binary string in single quotes enclosed in parentheses, as the following example illustrates:

```
USE Master;
GO
CREATE ENDPOINT DagobahEPv6
STATE = STARTED
AS TCP
```

```
    (LISTENER_PORT = 50102
   , LISTENER_IP = ('fe80::846a:46a7:b245:5255'))
FOR TSQL();
GO
```

In a previous example, the TCP/IP protocol was configured for the named instance DAGOBAH to listen on port 50101. With the additional endpoint and associated port, it will be necessary to add the port to the TCP/IP protocol with SQL Server Configuration Manager. This is done by simply adding another port to the port assignment delimited by a comma, as shown in Figure 7-5.



IP6	
Active	Yes
Enabled	No
IP Address	2001:0:4137:9e50:102b:26da:53
TCP Dynamic Ports	0
TCP Port	
IP7	
Active	Yes
Enabled	No
IP Address	fe80::102b:26da:53ef:e566%11
TCP Dynamic Ports	0
TCP Port	
IPAll	
TCP Dynamic Ports	
TCP Port	50501,50502

Active
Indicates whether the selected IP Address is active.

Figure 7-5: Adding another port to the port assignment.

Database Mirroring Endpoints

SQL Server 2008 uses a mirroring endpoint for exclusive use of the server that is configured to participate in a database mirroring configuration. In mirroring, which is described in Chapter 12, each instance of SQL Server is required to have its own dedicated database mirroring endpoint. All mirroring communication uses this database mirroring endpoint, but client connections to a database configured with a mirror use the standard TDS endpoint.

The configuration of an exclusive mirroring endpoint ensures that database mirror process communication is handled in a separate process from all other database activities. The easiest way to configure mirroring endpoints is to run the Mirroring Wizard as explained in Chapter 12. To create and configure a mirroring endpoint manually and enforce secure encrypted communication over the endpoint, the following code can be used:

```
CREATE ENDPOINT AughtEightDagobahMirror
  AUTHORIZATION sa
  STATE=STARTED
```

```
AS TCP (LISTENER_PORT = 5022, LISTENER_IP = ALL)
FOR DATA_MIRRORING
(ROLE = PARTNER, AUTHENTICATION = WINDOWS NEGOTIATE
,ENCRYPTION = REQUIRED ALGORITHM RC4);
```

This example can be used to create the mirroring endpoint on either the principal or mirror server. It assumes that the same domain account is used for the SQL Server service on both the principal and the mirror. For the witness server, the `ROLE` argument would need to be changed to `WITNESS`.

If different accounts are used for each `MSSQLSERVER` service on the servers, logins that are mapped to the service accounts from each server will need to be granted the `CONNECT` permission to the other servers participating in the mirroring configuration. The following script can be run to ensure encrypted authenticated communication between the three servers configured to take part in a mirroring relationship. AughtEight is the principal server, Dagobah is the mirror server, and Tatooine is the witness server. In this example, all three instances are running on the same physical server, which is why each endpoint is configured with a different port number. In the case of separate physical servers, the port numbers could be configured consistently.

```
USE Master;
GO
CREATE ENDPOINT AughtEightDagobahPrincipal
 AS TCP (LISTENER_PORT = 5022)
 FOR DATA_MIRRORING (ROLE = PARTNER, ENCRYPTION = REQUIRED ALGORITHM RC4);
GO
CREATE LOGIN [AughtEight\DagobahSQL] FROM WINDOWS;
CREATE LOGIN [AughtEight\TatooineSQL] FROM WINDOWS;
GO
GRANT CONNECT ON ENDPOINT::AughtEightDagobahPrincipal
 TO [AughtEight\TatooineSQL];
GRANT CONNECT ON ENDPOINT::AughtEightDagobahPrincipal
 TO [AughtEight\DagobahSQL];

--Run on Dagobah
USE Master;
GO
CREATE ENDPOINT AughtEightDagobahMirror
  AS TCP (LISTENER_PORT = 5023)
  FOR DATA_MIRRORING (ROLE = PARTNER, ENCRYPTION = REQUIRED ALGORITHM RC4);
GO
CREATE LOGIN [AughtEight\AughtEightSQL] FROM WINDOWS;
CREATE LOGIN [AughtEight\TatooineSQL] FROM WINDOWS;
GO
GRANT CONNECT ON ENDPOINT::AughtEightDagobahMirror
TO [AughtEight\AughtEightSQL];
GRANT CONNECT ON ENDPOINT::AughtEightDagobahMirror
TO [AughtEight\TatooineSQL];

--Run on Tatooine
USE Master;
GO
CREATE ENDPOINT AughtEightDagobahWitness
  AS TCP (LISTENER_PORT = 5024)
  FOR DATA_MIRRORING (ROLE = WITNESS, ENCRYPTION = REQUIRED ALGORITHM RC4);
GO
CREATE LOGIN [AughtEight\AughtEightSQL] FROM WINDOWS;
CREATE LOGIN [AughtEight\DagobahSQL] FROM WINDOWS;
GO
GRANT CONNECT ON ENDPOINT::AughtEightDagobahWitness
TO [AughtEight\AughtEightSQL];
GRANT CONNECT ON ENDPOINT::AughtEightDagobahWitness
TO [AughtEight\DagobahSQL];
```

The preceding commands set up the communication framework for mirroring, but do not actually initialize mirroring. See Chapter 12 for more information on how to configure and monitor mirroring.

SOAP Endpoints

Simple Object Access Protocol (SOAP) is a platform-independent protocol that defines how to use XML and HTTP to access services, objects, and servers. SOAP endpoints are created to publish SQL Server programming objects over data-tier Web Services without the use of IIS as a Web server.

Data-tier Web Services provide a very powerful alternative to XML Web Services and provide the means of exposing stored procedures and functions over HTTP the same as conventional Web Service architecture. In addition to stored procedures and functions, SOAP endpoints can be configured to allow ad hoc queries, but as a general rule, ad hoc access should be avoided.

Although SOAP endpoints were introduced in SQL Server 2005, they have been considered deprecated in SQL Server 2008. Microsoft intends to remove SOAP endpoints in a future version of SQL Server. Microsoft recommends that existing applications that use SOAP endpoints should instead switch to using either ASP.NET or Windows Communications Foundation (WCF). The information provided in this section regarding the configuration of SOAP endpoints should be treated as reference only, and development of applications that use SOAP endpoints in SQL Server should be discouraged.

SOAP endpoints return SOAP messages consisting of an XML document with a header and a body. SOAP messages are essentially one-way transmissions from a sender to a receiver. SOAP does not define any application semantics such as a programming model or implementation-specific details. Web Services, on the other hand, require a request/response model. The solution is to send SOAP messages within the body of an HTTP request and response. This solution provides the required model for Web Services, and SOAP endpoints provide the structure to accomplish the communication.

In order to be able to create SOAP endpoints successfully, the URL must already be registered with the HTTP.sys kernel-mode driver. This will ensure that requests for the SOAP endpoint URL are passed to the SQL Server, and not handled by other applications, such as IIS. If the SQL Server service account has full administrative permissions on the local machine, this can be performed by using the `sp_reserve_http_namespace` stored procedure. However, since it is generally not recommended to run SQL Server services under an administrative account, you may have to register the URL manually using the `netsh` Windows configuration tool (for Windows Server 2003 environments, use the `httpcfg` Windows system tool) before executing the `sp_reserve_http_namespace` stored procedure.

The following syntax is used for creating a SOAP endpoint:

```
CREATE ENDPOINT endPointName [ AUTHORIZATION login ]
STATE = { STARTED | STOPPED | DISABLED }
AS HTTP (
PATH = 'url'
  , AUTHENTICATION =( { BASIC | DIGEST | INTEGRATED | NTLM | KERBEROS }
    [ ,...n ] )
  , PORTS = ( { CLEAR | SSL} [ ,... n ] )
   )

FOR SOAP (
   [ { WEBMETHOD ['namespace'.] 'method_alias'
   (   NAME = 'database.schema.name'
   )
   } [ ,...n ] ]
   [ , DATABASE = { 'database_name' | DEFAULT }
   [ , HEADER_LIMIT = int ])
```

Because syntax specifications can be a bit arcane, the following example is provided to demonstrate the creation of a SOAP endpoint. The example creates a SOAP endpoint called *AWSales* that uses Windows integrated security to control access to a Web Service that is published at the location `http://AughtEight/AdventureWorks2008/Sales`. The endpoint exposes the stored procedure `AdventureWorks2008.dbo.uspGetBillOfMaterials` as the Web method `GetBillOfMaterials`. The SOAP document that is created by this Web Service can be viewed by opening Internet Explorer and navigating to the Web Service URL and appending a Web Service Description Language (WSDL) query to the end of the URL:

```
http://AughtEight/AdventureWorks2008/Sales?wsdl
```

Keep in mind that you will most likely have to change the server name in your environment.

```
USE master;
--Turn on Advanced Options for the sp_configure utility
EXEC sp_configure 'show advanced option', '1';
RECONFIGURE
GO
-- enable xp_cmdshell
EXEC sp_configure 'xp_cmdshell', '1';
RECONFIGURE
GO
--Allow SQL to create the reservation (Windows Server 2008/Vista only)
EXEC xp_cmdshell 'netsh http add urlacl url=
http://AughtEight:80/AdventureWorks2008 user=AughtEight\SQLService delegate=yes';
GO

--Reserve the URL
EXEC sp_reserve_http_namespace N'http://AughtEight:80/AdventureWorks2008';
GO
```

```
-- Create the SOAP endpoint
USE Master;
GO
CREATE ENDPOINT AWSales
STATE = STARTED
AS HTTP(
        PATH = '/AdventureWorks2008/Sales'
       ,AUTHENTICATION = (INTEGRATED)
       ,PORTS = ( CLEAR )
       ,SITE = 'AughtEight')
FOR SOAP(
WEBMETHOD 'GetBillOfMaterials'
       (NAME='AdventureWorks2008.dbo.uspGetBillOfMaterials'
      ,FORMAT=ROWSETS_ONLY)
      ,WSDL = DEFAULT
      ,DATABASE = 'AdventureWorks2008'
      ,NAMESPACE = 'http://AughtEight/'
   );
GO
```

In the preceding example, the `xp_cmdshell` extended stored procedure was used to execute the necessary `netsh` command to allow SQL to register the URL, although this can be just as easily done from a command line. Because `xp_cmdshell` is disabled by default, the `sp_configure` stored procedure had to be run to enable its use.

Although Internet Explorer can be used to view the SOAP document, the real use for data-tier Web Services is for applications that are created to connect to and consume XML Web Services. Later in this chapter, you will see how to do this.

The HTTP arguments available for configuration in a `CREATE ENDPOINT` statement are described in the following table:

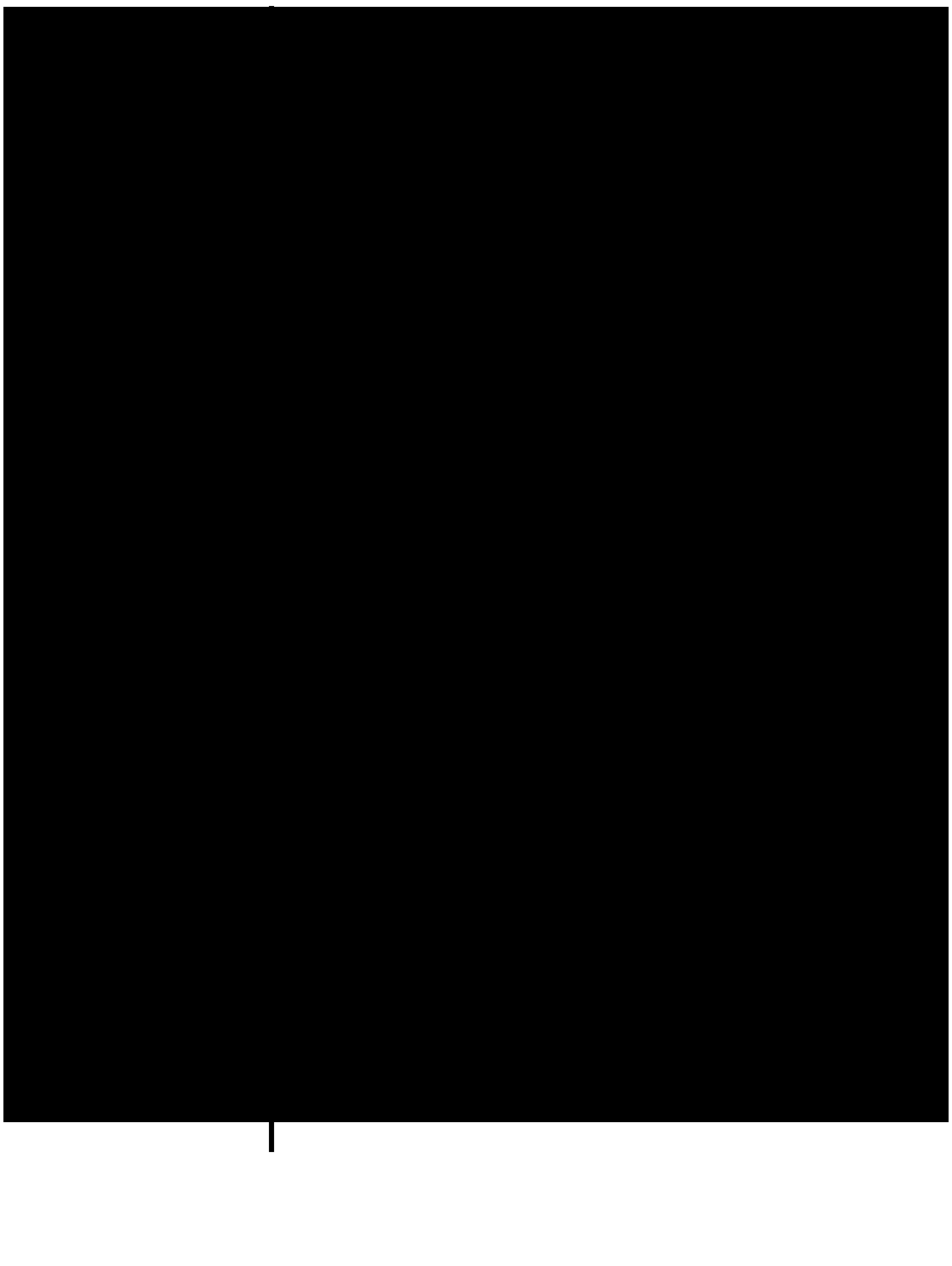
[redacted]

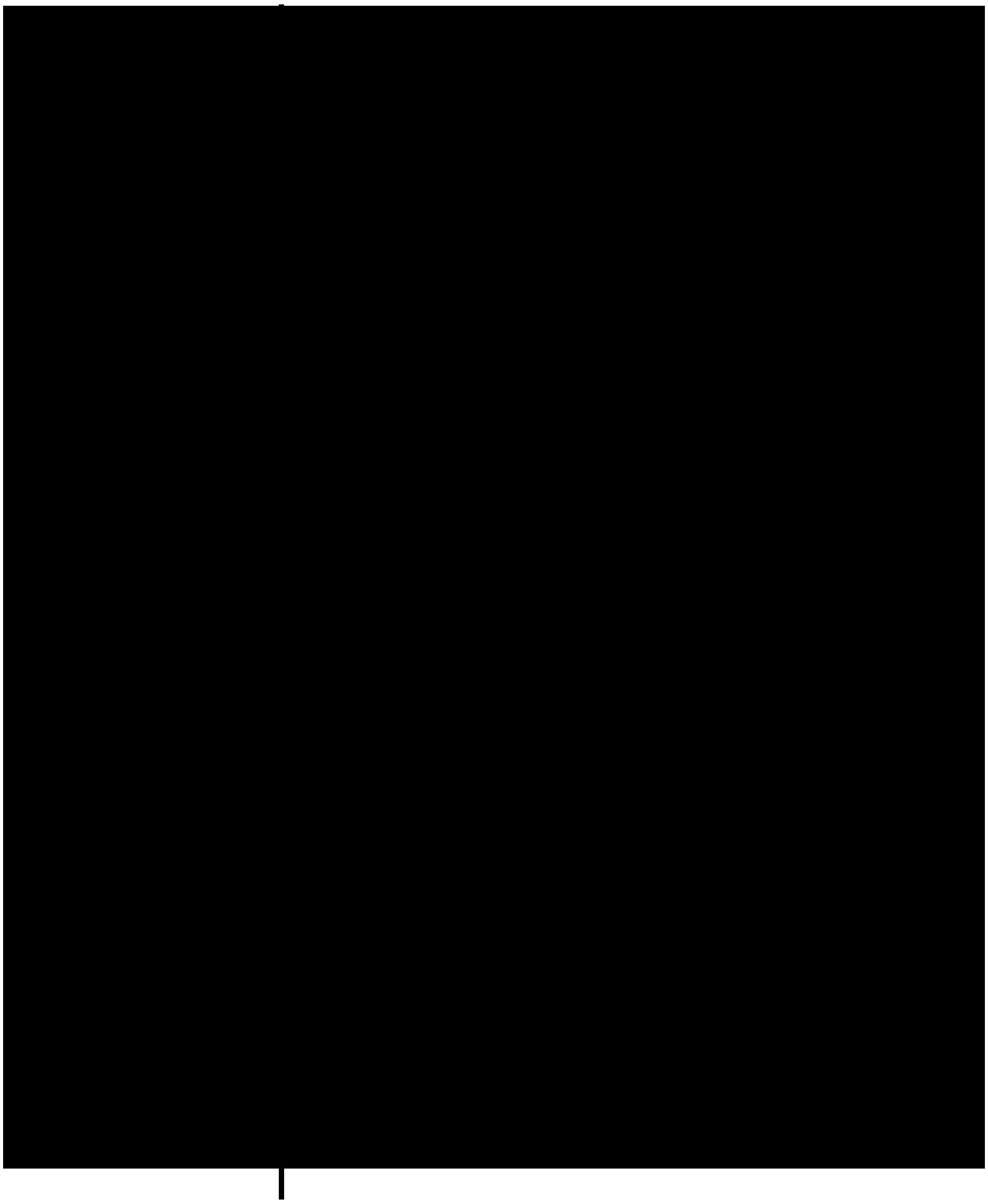
[redacted]

The configurable SOAP arguments are described in the following table:

[redacted]

Continued





Service Broker Endpoints

As described in Chapter 19, Service Broker is a powerful feature of SQL Server 2008 that enables database applications to communicate asynchronously with other database applications in a Service-Oriented Architecture (SOA). Service Broker endpoints are only required if the two instances of the broker service are located on separate instances of SQL Server. They are created in much the same way as SOAP endpoints. The basic `CREATE ENDPOINT` command is used, but instead of the `FOR SOAP` clause that defines the endpoint as a SOAP endpoint, the `FOR SERVICE_BROKER` clause is used. The syntax for creating a Service Broker endpoint is as follows:

```
CREATE ENDPOINT endPointName [ AUTHORIZATION login ]
STATE = { STARTED | STOPPED | DISABLED }
AS TCP (
  LISTENER_PORT = listenerPort
  [ [ , ] LISTENER_IP = ALL | ( 4-part-ip ) | ( "ip_address_v6" ) ]
)
FOR SERVICE_BROKER (
   [ AUTHENTICATION = { WINDOWS [ { NTLM | KERBEROS | NEGOTIATE } ]
   | CERTIFICATE certificate_name
   | WINDOWS [ { NTLM | KERBEROS | NEGOTIATE } ] CERTIFICATE certificate_name
   | CERTIFICATE certificate_name WINDOWS [ { NTLM | KERBEROS | NEGOTIATE } ]
} ]
   [ [ , ] ENCRYPTION = { DISABLED | { { SUPPORTED | REQUIRED }
       [ ALGORITHM { RC4 | AES | AES RC4 | RC4 AES } ] }
   ]
   [ [ , ] MESSAGE_FORWARD_SIZE = forward_size ]
)
```

An example of this syntax put in use to create a Service Broker endpoint is as follows:

```
CREATE ENDPOINT MyEndpoint
STATE = STARTED
AS TCP ( LISTENER_PORT = 50001 )
FOR SERVICE_BROKER ( AUTHENTICATION = WINDOWS );
```

In all likelihood, when creating Service Broker or mirroring endpoints, certificates will be used to ensure authenticated and encrypted traffic between endpoints, especially if the endpoints are located on different physical servers. For more information on the workings of Service Broker, take a look at Chapter 19. For information about creating and using certificates, see Chapter 6.

Securing Endpoints

A critically important aspect of all endpoints is securing them so that only connections that are authorized can enumerate and call the Web methods or other services that the endpoint provides. The key permission for endpoints is the `CONNECT` permission. Only those logins that have been explicitly granted the `CONNECT` permission will be able to expose the functionality behind the endpoint. In addition, the login will need permissions to the underlying objects that the endpoint provides access for.

Data-Tier Web Services

To re-create this exercise requires that you have Visual Studio 2008 installed, as well as SQL Server 2008. As described in Chapter 2, the SQL Server 2008 installation installs a piece of Visual Studio, but it does not install everything you need to create database applications. The following examples and descriptions assume that you have installed either C# or VB.NET (or both). If you haven't, the information is still very useful, but you will not be able to practice or re-create it. The examples using Visual Studio may seem to be a bit out of context in this book. However, it is difficult to describe the use of SOAP endpoints without using a Visual Studio application to demonstrate the purpose of data tier Web Services.

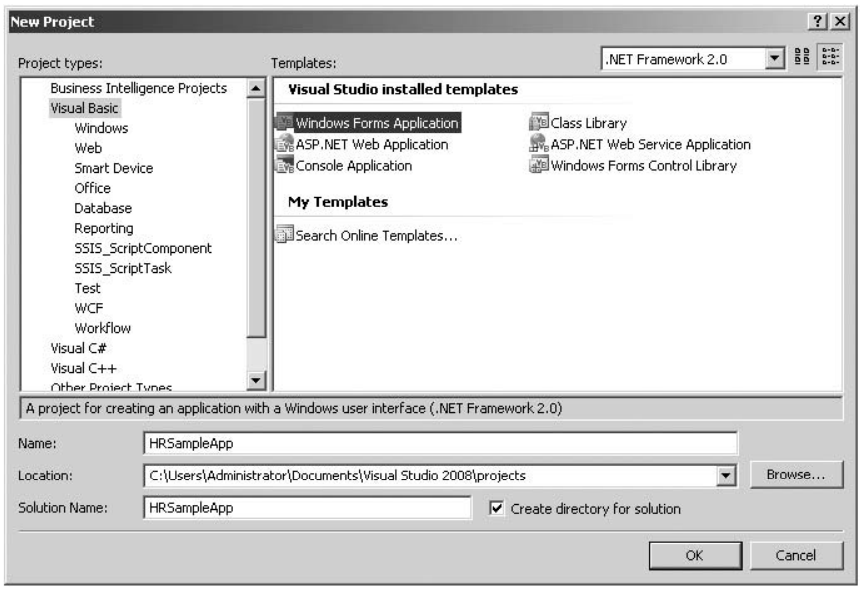
Create the Endpoint

The first step is to create the endpoint that will publish the two stored procedures you want to make available via a data-tier Web Service where they can be used by any SOAP-compliant application. Execute the following code to create the SOAP endpoint `HRWebService` that publishes the `uspGetEmployeeManagers` and `uspGetManagerEmployees` stored procedures as the `GetEmployeeManagers` and `GetManagerEmployees` Web methods:

```
USE Master;
GO
CREATE ENDPOINT HRWebService
STATE = STARTED
AS HTTP(
        PATH = '/AdventureWorks2008/HR'
       ,AUTHENTICATION = (INTEGRATED)
       ,PORTS = ( CLEAR )
       ,SITE = 'AughtEight')
FOR SOAP(
         WEBMETHOD 'GetEmployeeManagers'
           (NAME='AdventureWorks2008.dbo.uspGetEmployeeManagers'
           ,FORMAT=ROWSETS_ONLY)
        ,WEBMETHOD 'GetManagerEmployees'
           (NAME='AdventureWorks2008.dbo.uspGetManagerEmployees'
           ,FORMAT=ROWSETS_ONLY)
        ,WSDL = DEFAULT
        ,DATABASE = 'AdventureWorks2008'
        ,NAMESPACE = 'http://AughtEight/'
   );
GO
```

Once the endpoint has been created to make the procedures visible through the Web Service, a SOAP-compliant application will be able to enumerate and reference the Web methods specified in the endpoint.

1. Start Visual Studio 2008 and create a new VB.NET or C# Windows Application Project by clicking on the File menu, selecting New, and then Project.
2. In the New Project window, select either Visual Basic or Visual C# from the Project Types pane, and then choose Windows Forms Application from the Templates pane, as shown in Figure 7-6. Ensure that .NET Framework 2.0 is selected (this will not work with later versions of .NET).



A project for creating an application with a Windows user interface (.NET Framework 2.0)

Name: HRSampleApp

Location: C:\Users\Administrator\Documents\Visual Studio 2008\projects

Solution Name: HRSampleApp

☑ Create directory for solution

Figure 7-6: Selecting ''Windows Forms Application'' from the Templates pane.

3. Give the project a name such as *HRSampleApp*. Choose a folder for the solution to be created in, and click OK. A design window showing a blank Windows form will appear.
4. From the toolbox (to the left of the form designer by default), select and drag a button control to the upper-left-hand side of the form. Then drag a textbox and place it to the right of the button. Lastly, drag a `datagridview` control onto the form, and place it under the button and textbox controls, as shown in Figure 7-7. If the toolbox is not visible, it can be launched by pressing *[Ctrl]+[Alt]+X* or by selecting it from the View menu.
5. You may want to adjust the size of the form and the `datagridview` control to accommodate multiple columns and rows. After creating the form, right-click on the project name in the Solution Explorer window, and select ''Add Web Reference,'' as shown in Figure 7-8.

   The Add Web Reference window will display where the data-tier Web Service can be added as a Web reference to the project.
6. In the URL dropdown textbox, type in the appropriate address for your server followed by a WSDL query command. In my case, the URL and query take the form of `http://AughtEight/Adventureworks2008/hr?wsdl`.
7. Click the GO button to query the SQL Server for information regarding any Web methods published at that location. You should see results similar to those shown in Figure 7-9.
8. In the ''Web reference name'' field, type in the name **HRWebService**, and click OK.

Now that all the foundation work has been completed, it is time to write the code that will call on the Web methods made available with the SOAP endpoint.

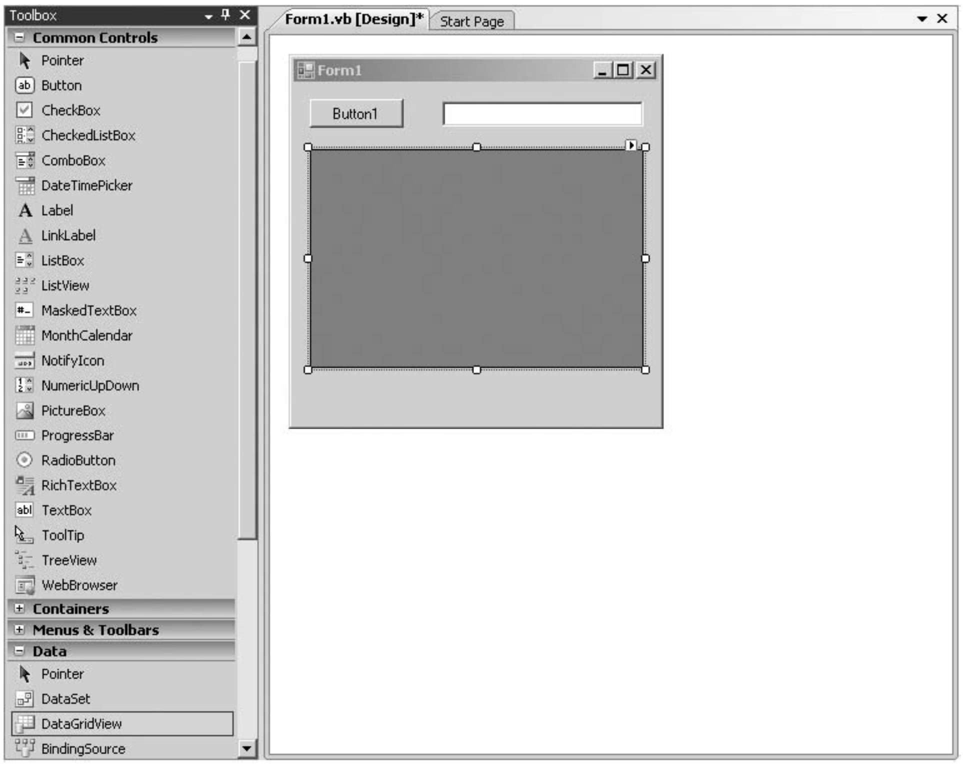


Figure 7-7: Placing a `datagridview` control.

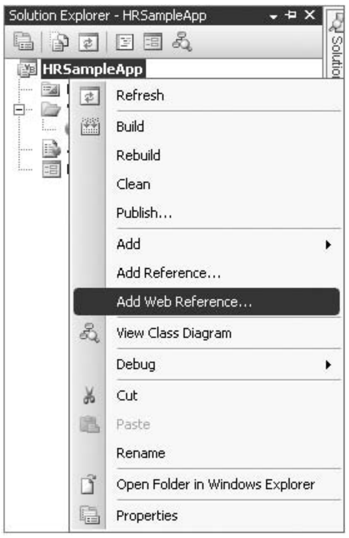
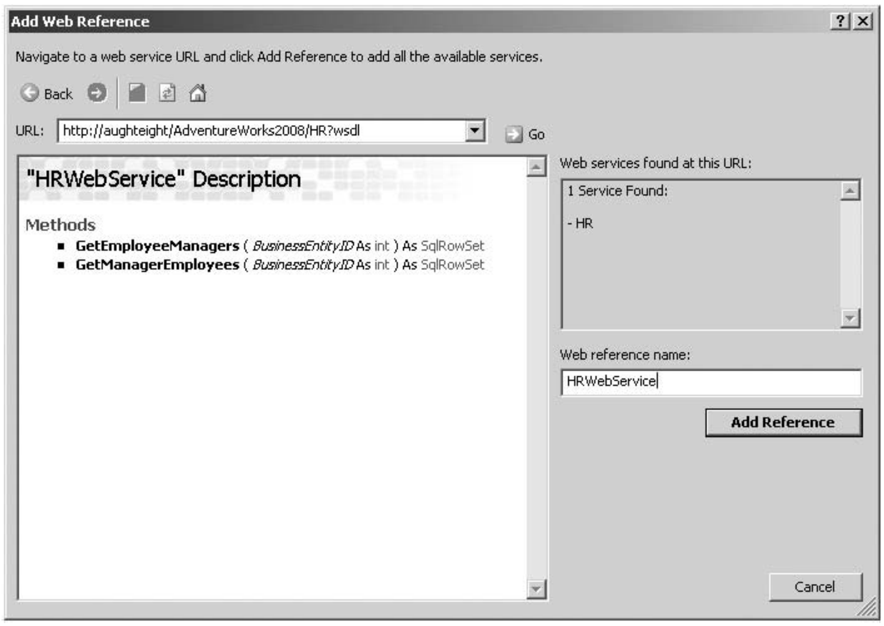


Figure 7-8: Selecting "Add Web Reference."


Add Web Reference

Navigate to a web service URL and click Add Reference to add all the available services.

URL: http://aughteight/AdventureWorks2008/HR?wsdl

"HRWebService" Description

Methods
- **GetEmployeeManagers** (*BusinessEntityID* As int) As SqlRowSet
- **GetManagerEmployees** (*BusinessEntityID* As int) As SqlRowSet

Web services found at this URL:

1 Service Found:

- HR

Web reference name: HRWebService

Figure 7-9: Viewing the results of a query for information regarding Web methods.

9. Double-click on the Button1 button on the Form Designer. This will launch the Code Editor window and create the basic code to handle the button `click` event. In the button `click` event handler, type in the code shown in the next example. There is one set of code for a VB.NET application, and another for Visual C# application.

Following is the Visual C# code:

```
private void button1_Click(object sender, EventArgs e)
{
  DataSet dsEmployees;
            HRWebService.HRWebService proxy =
                new HRSampleApp.HRWebService.HRWebService();
            proxy.Credentials = System.Net.CredentialCache.DefaultCredentials;

            try
            {
                Int32 intMgrID;
                intMgrID = Convert.ToInt32(textBox1.Text);
                dsEmployees = proxy.GetManagerEmployees(intMgrID);
                dataGridView1.DataSource = dsEmployees;
                dataGridView1.DataMember = dsEmployees.Tables[0].TableName;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
   }
}
```

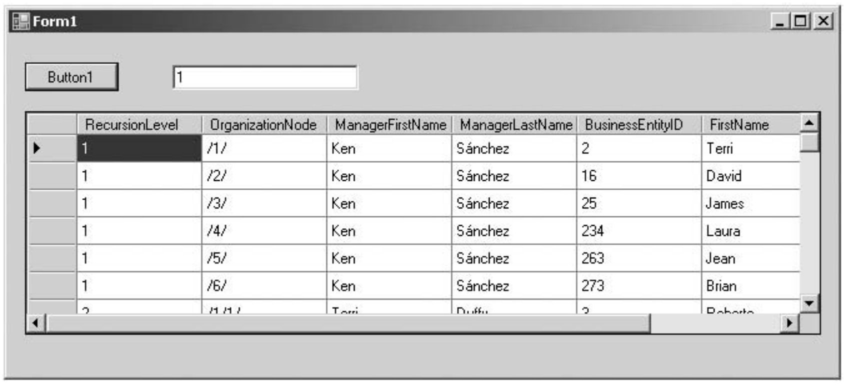
Following is the Visual Basic.NET code:

```
Private Sub button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnGetEmployees.Click

  Dim Proxy As New HRWebService.HRWebService
  Proxy.Credentials = System.Net.CredentialCache.DefaultCredentials

   Try

     Dim dsEmployees As DataSet = Proxy.GetManagerEmployees(textBox1.Text)
     dataGridView1.DataSource = dsEmployees
     dataGridView1.DataMember = dsEmployees.Tables(0).TableName

   Catch
     MsgBox(Err.Description)
   End Try

End Sub
```

Notice that the amount of code required to consume the Web Service is actually very small. Not counting error-handling code, there are only five lines of code for VB.NET and eight lines for Visual C#. This is one of the features that make consuming Web Services so attractive; most of the work has been done at the Web Service side.

10. Once the code has been entered into the button `click` event, press [F5] or click the green triangle on the menu to start the application debugging process. If everything goes well, what you should see is the Windows form created earlier.

11. Enter the number **1** in the textbox, and click Button1. Your results should look like those in Figure 7-10.



RecursionLevel	OrganizationNode	ManagerFirstName	ManagerLastName	BusinessEntityID	FirstName
1	/1/	Ken	Sánchez	2	Terri
1	/2/	Ken	Sánchez	16	David
1	/3/	Ken	Sánchez	25	James
1	/4/	Ken	Sánchez	234	Laura
1	/5/	Ken	Sánchez	263	Jean
1	/6/	Ken	Sánchez	273	Brian
2	/1/1/	Terri	Duffy	3	Roberto

Figure 7-10: Results of entering *1*.

SOAP endpoints can be created to not only return data, but also to manipulate data in the database. The amount of code required does not change dramatically.

As a database administrator, this may all seem a bit over the top, but it is very important to understand why developers may want to use SOAP endpoints and exactly what they do. Keep in mind, however, that SOAP endpoints are no longer supported in SQL Server 2008. Instead, encourage your developers to use Windows Communications Foundation in the newer versions of .NET Framework.

Summary

SQL Configuration Manager offers the database administrator a one-stop shop for troubleshooting and configuring SQL Server connection objects and networking devices. The tools you will use to manage these objects are simple, if not intuitive. Diagnosing networking problems has never been easier. Using the information in this chapter, you should be able to configure and secure the network protocols and endpoints that make it possible to make the most of SQL Server 2008 services and features. With the introduction of Service Broker and mirroring, the database administrator's responsibility for network and transport security has never been greater. Be sure to carefully evaluate all the security and configuration options available for each networking object to ensure the highest level of security and functionality.

In Chapter 8, you will learn about automating SQL Server 2008 administrative and maintenance processes. You'll learn to configure jobs and alerts that will keep you informed of SQL Server performance, and keep it performing at peak efficiency.