

SOA Support in .NET

The .NET framework is a proprietary solution runtime and development platform designed for use with Windows operating systems and server products. The .NET platform can be used to deliver a variety of applications, ranging from desktop and mobile systems to distributed Web solutions and Web services.

A primary part of .NET relevant to SOA is the ASP.NET environment, used to deliver the Web Technology layer within SOA (and further supplemented by the Web Services Enhancements (WSE) extension).

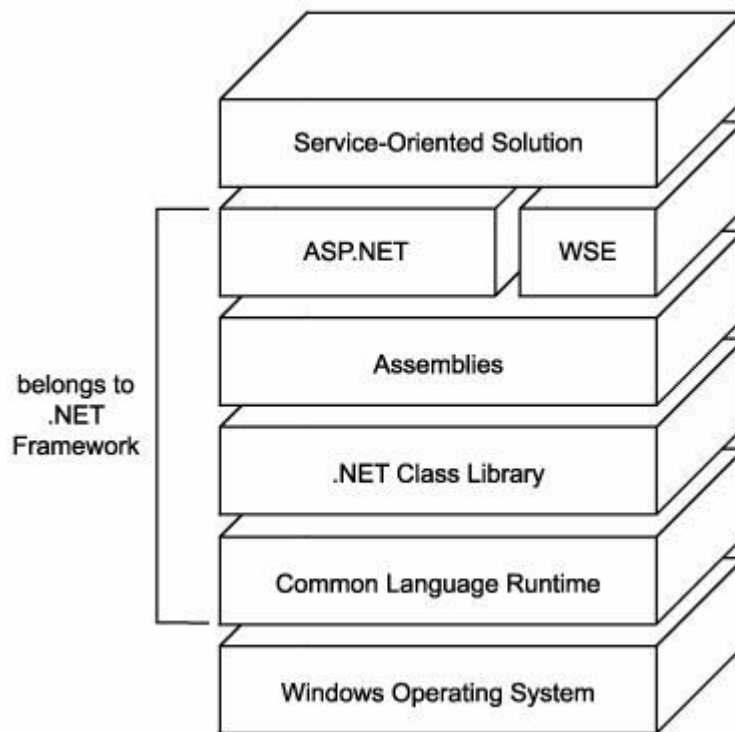


Figure 1: SOA support in .NET

API's in .NET for SOA

.NET provides programmatic access to numerous framework (operating system) level functions via the .NET Class Library, a large set of APIs organized into namespaces. Each namespace must be explicitly referenced for application programming logic to utilize its underlying features.

Following are examples of the primary namespaces that provide APIs relevant to Web services development:

- System.Xml Parsing and processing functions related to XML documents are provided by this collection of classes. Examples include:

- The `XmlReader` and `XmlWriter` classes that provide functionality for retrieving and generating XML document content.
 - Fine-grained classes that represent specific parts of XML documents, such as the `XmlNode`, `XmlElement`, and `XmlAttribute` classes.
- **System.Web.Services** This library contains a family of classes that break down the various documents that comprise and support the Web service interface and interaction layer on the Web server into more granular classes. For example:
 - WSDL documents are represented by a series of classes that fall under the `System.Web.Services.Description` namespace.
 - Communication protocol-related functionality (including SOAP message documents) are expressed through a number of classes as part of the `System.Web.Services.Protocols` namespace.
 - The parent `System.Web.Services` class that establishes the root namespace also represents a set of classes that express the primary parts of ASP.NET Web service objects (most notably, the `System.Web.Services.WebService` class).
 - Also worth noting is the `SoapHeader` class provided by the `System.Web.Services.Protocols` namespace, which allows for the processing of standard SOAP header blocks.

In support of Web services and related XML document processing, a number of additional namespaces provide class families, including:

- `System.Xml.Xsl` Supplies documentation transformation functions via classes that expose XSLT-compliant features.
- `System.Xml.Schema` A set of classes that represent XML Schema Definition Language (XSD)-compliant features.
- `System.Web.Services.Discovery` Allows for the programmatic discovery of Web service metadata.

Service Providers in .NET

.NET service providers are Web services that exist as a special variation of ASP.NET applications, called ASP.NET Web Services. You can recognize a URL pointing to an ASP.NET Web Service by the ".asmx" extension used to identify the part of the service that acts as the endpoint. ASP.NET Web Services can exist solely of an ASMX file containing inline code and special directives, but they are more commonly comprised of an ASMX endpoint and a compiled assembly separately housing the business logic. Figure on the next page shows how the pieces of a .NET Web service are positioned within our service provider model.

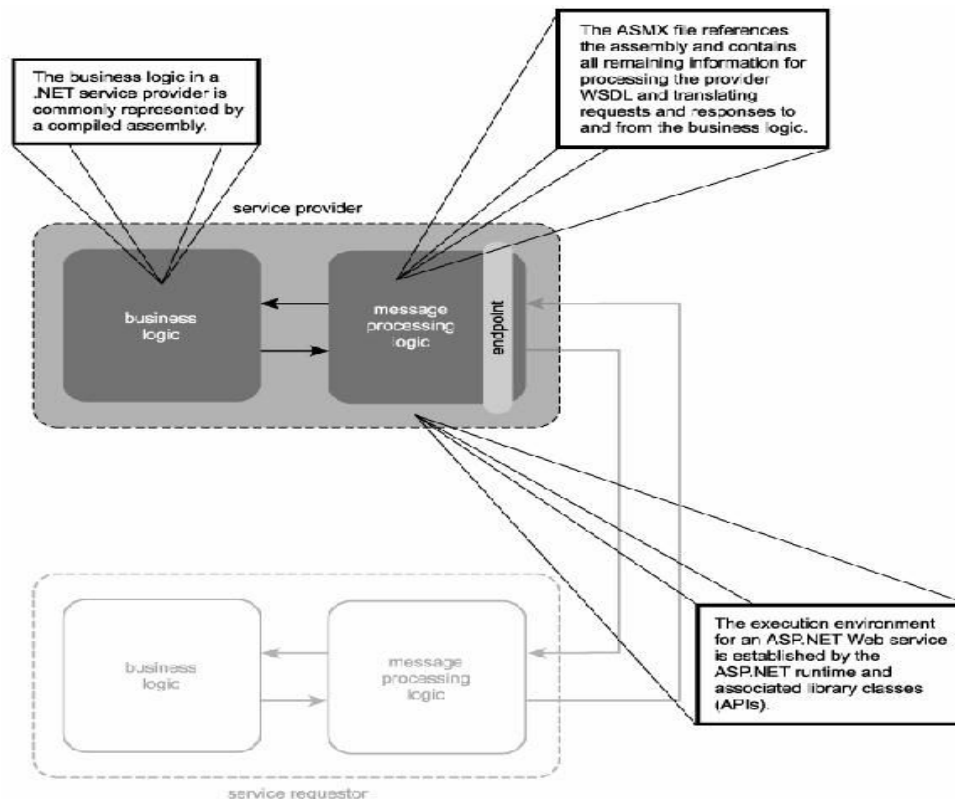


Figure 2: Service Provider Model in .NET

Service requestors

To support the creation of service requestors, .NET provides a proxy class that resides alongside the service requestor's application logic and duplicates the service provider interface. This allows the service requestor to interact with the proxy class locally, while delegating all remote processing and message marshalling activities to the proxy logic.

The .NET proxy translates method calls into HTTP requests and subsequently converts the response messages issued by the service provider back into native method return calls.

The code behind a proxy class is auto-generated using Visual Studio or the WSDL.exe command line utility. Either option derives the class interface from the service provider WSDL definition and then compiles the proxy class into a DLL. Figure explains how .NET proxies behave within the standard service request model

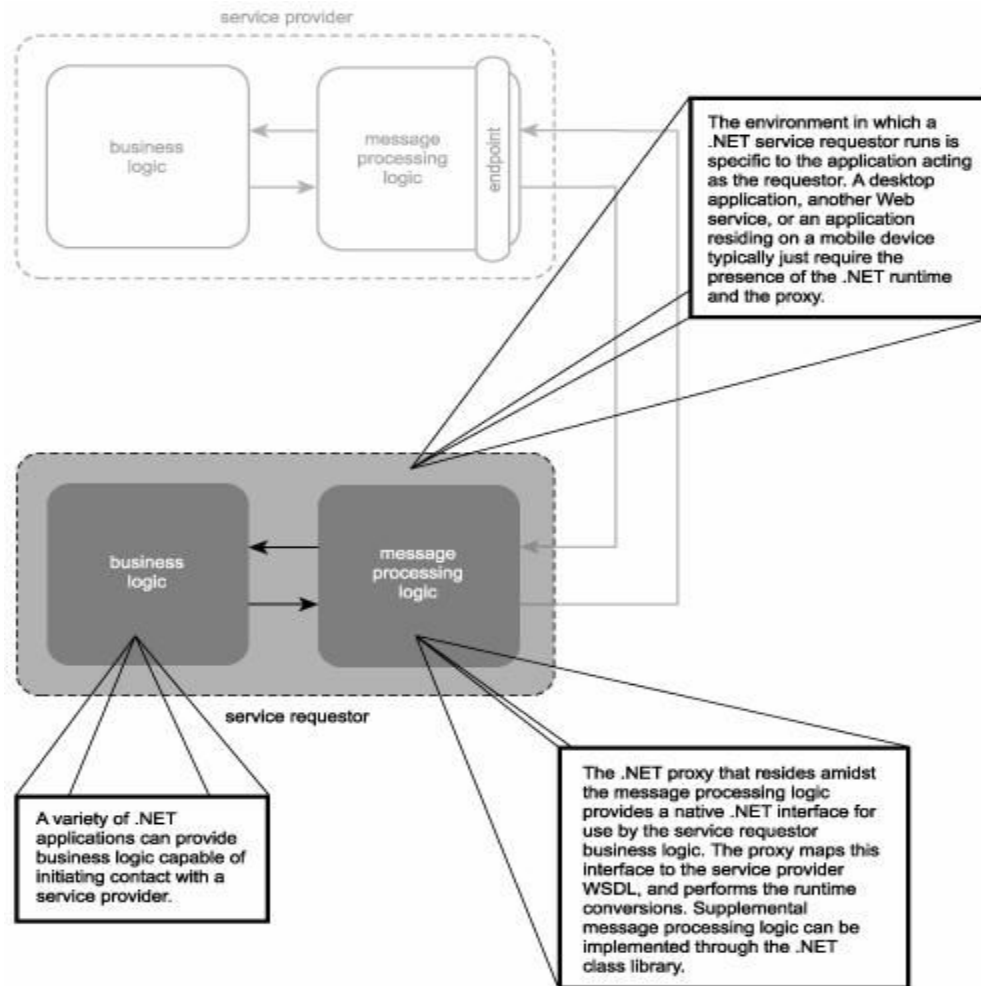


Figure 3: Service Request Model

Service agents

The ASP.NET environment utilizes many system-level agents that perform various runtime processing tasks. As mentioned earlier, the ASP.NET runtime outfits the HTTP Pipeline with a series of HTTP Modules. These service agents are capable of performing system tasks such as authentication, authorization, and state management. Custom HTTP Modules also can be created to perform various processing tasks prior and subsequent to endpoint contact. Figure Shows .Net Service Agents

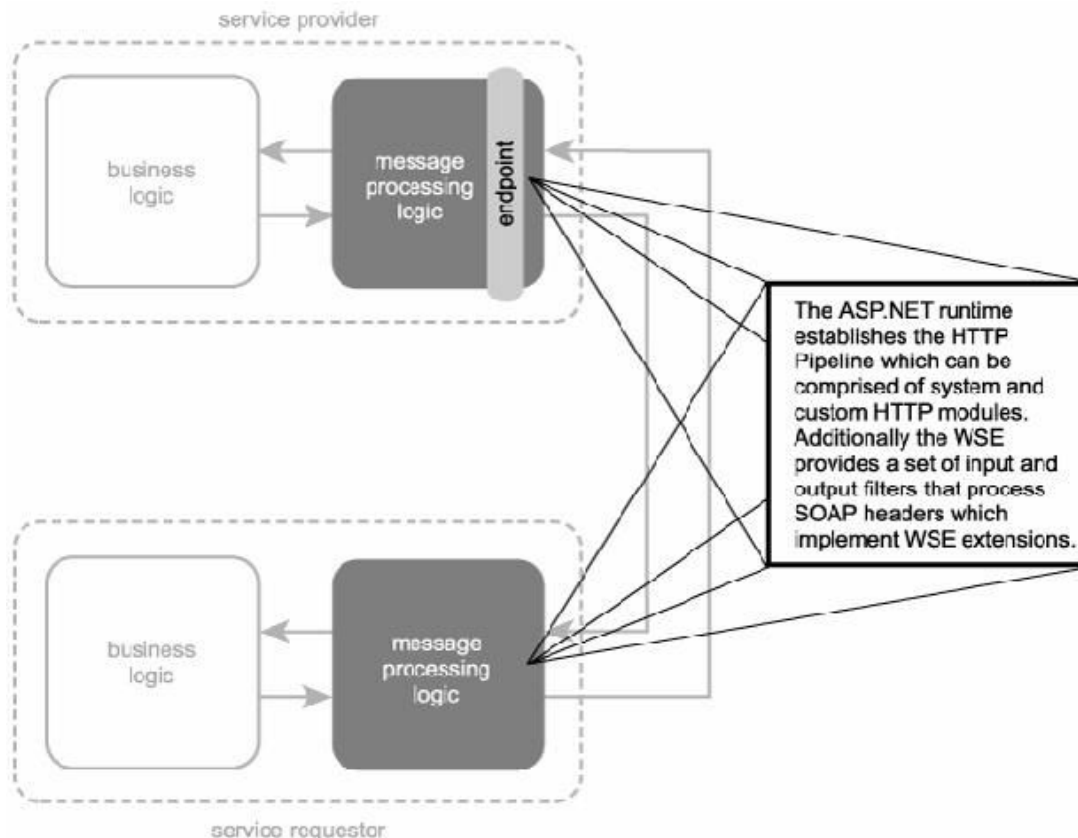


Figure 4: Service Agents in .NET

Also worth noting are HTTP Handlers, which primarily are responsible for acting as runtime endpoints that provide request processing according to message type. As with HTTP Modules, HTTP Handlers can also be customized. (Other parts of the HTTP Pipeline not discussed here include the HTTP Context, HTTP Runtime, and HTTP Application components.)

Another example of service agents used to process SOAP headers are the filter agents provided by the WSE toolkit (officially called WSE filters). The feature set of the WSE is explained in the next section ([Platform extensions](#)), but let's first briefly discuss how these extensions exist as service agents.

Support for service-orientation principles

The four principles we identified earlier as being those not automatically provided by first-generation Web services technologies are the focus of this section, as we briefly highlight relevant parts of the .NET framework that directly or indirectly provide support for their fulfillment.

Autonomy

The .NET framework supports the creation of autonomous services to whatever extent the underlying logic permits it. When Web services are required to encapsulate application logic already residing in existing legacy COM components or assemblies designed as part of a traditional distributed solution, acquiring explicit functional boundaries and self-containment may be difficult.

However, building autonomous ASP.NET Web Services is achieved more easily when creating a new service-oriented solution, as the supporting application logic can be designed to support autonomy requirements. Further, self-contained ASP.NET Web Services that do not share processing logic with other assemblies are naturally autonomous, as they are in complete control of their logic and immediate runtime environments.

Reusability

As with autonomy, reusability is a characteristic that is easier to achieve when designing the Web service application logic from the ground up. Encapsulating legacy logic or even exposing entire applications through a service interface can facilitate reuse to whatever extent the underlying logic permits it. Therefore, reuse can be built more easily into ASP.NET Web Services and any supporting assemblies when developing services as part of newer solutions.

Statelessness

ASP.NET Web Services are stateless by default, but it is possible to create stateful variations. By setting an attribute on the service operation (referred to as the WebMethod) called `EnableSession`, the ASP.NET worker process creates an `HttpSessionState` object when that operation is invoked. State management therefore is permitted, and it is up to the service designer to use the session object only when necessary so that statelessness is continually emphasized.

Discoverability

Making services more discoverable is achieved through proper service endpoint design. Because WSDL definitions can be customized and used as the starting point of an ASP.NET Web Service, discoverability can be addressed, as follows:

- The programmatic discovery of service descriptions and XSD schemas is supported through the classes that reside in the `System.Web.Services.Discovery` namespace. The .NET framework also provides a separate UDDI SDK.

- .NET allows for a separate metadata pointer file to be published alongside Web services, based on the proprietary DISCO file format. This approach to discovery is further supported via the Disco.exe command line tool, typically used for locating and discovering services within a server environment.
- A UDDI Services extension is offered on newer releases of the Windows Server product, allowing for the creation of private registries.
- Also worth noting is that Visual Studio contains built-in UDDI support used primarily when adding services to development projects.