

---

# Web Services

---

## Core Web Services Process Functionality

## Introduction

What are Web Services? There certainly is a lot of hype around this technology and some of it is, for once, justified. Web Services encapsulate the process of contacting well-defined services over a network and invoking methods within those services using XML data. Those services respond with XML data. In addition the well-defined services can be registered in a specialised directory so that a service and its descriptor can be discovered by service type.

All of this is relatively standard to a distributed system that you would find in CORBA, DCOM, RMI, etc., except that Web Services have achieved platform, vendor and technology neutrality in the same way in which Web Page, in the early days, were completely agnostic about everything.

This means that a Web Service implemented in Visual C++ running under an IIS server should be usable from a Java client, as should a J2EE Web Service be available to a C# .NET client. In fact you shouldn't even care what platform or language a Web Service is implemented on, just as you don't really care what type of Web Server is handling the web site you are viewing.

From a software architect's point of view this is great, since they no longer have to follow the industry fights between Microsoft, IBM, Sun and the multitude of other big players. Finally a software architect can concentrate on the business concepts they are striving to deliver. For a software developer the benefits are enormous since they can implement the components on whatever platform they want without being restricted by paranoid strategic direction. Finally a base technology has started to unify a fractured and squabbling industry.

So is the hype justified? Well, some of it probably is, but remember that the basis of Web Services is probably a little overstated. Sending a structured XML message over a transport (HTTP, SMTP etc) is not that new or clever. Describing the structure of the message and the nature of the endpoint using another XML document is slightly more useful, especially if it is done in a neutral way. Creating client stub code automatically and converting it into executable objects is quite useful. Being able to register Web Services and allow potential clients to search for a service provides a superb disconnect from the Web Service client and the Web Service provider.

What is the relevance for Procession? Procession has an easy-to-use process engine that is almost unique in its ability to integrate human interactions into a workflow whilst maintaining complex, data-centric process capabilities. Providing the capability to integrate Procession process flows with Web Services is a real bonus since it removes many of the traditional integration headaches associated with proprietary protocols and data formats. Unlike IBM's Web Services Flow Language (WSFL), Microsoft's XLANG and BPMI (Intalio) Business Process Modelling Language (BPML), all of which integrate process and web services to varying extents, Procession's approach ensures integration with the business users whilst maintaining the advantages of back-office business process automation (BPA).

## Web Services, the basics

Web Services are generally accepted to consist of three components; the service provision, the service description, and a method of publishing and discovering the service. Service publishing and discovery is delivered via Universal Description Discovery and Integration (UDDI), itself a Web Service. Service description is delivered via Web Services Definition Language (WSDL). These two "protocols" form the basis for Web Services. The service that is described by a WSDL document only needs to conform to some very loose requirements. The service must be accessible over some form of network transport, and that's about it. Most web services use the HTTP protocol for transport. Additionally most web services encode their data using SOAP protocol techniques.

At its base level this means that a "normal" Web Service accepts request data as an XML document transported over HTTP, and responds synchronously with an XML document in response to that request. The variations on this theme are Web Services that accept normal HTTP POST and GET parameters, and those that respond with non-XML data. Alternatively the transport could be SMTP, in which case there

may be long delays between client request and any response being delivered back to the client. Asynchronous messaging is also a possibility, by simply not caring what the response is, through to using a Message Oriented Middleware (MOM) transport such as MQ Series.

So at the base level we send an XML Document to the Web Service and we expect the Web Service to reply with an XML Document. The document we send, the destination we send it to and the response we receive, will all conform to the specification in the WSDL document. If the service was published into a UDDI directory the response will deliver the service that was published to UDDI.

## Structure of SOAP

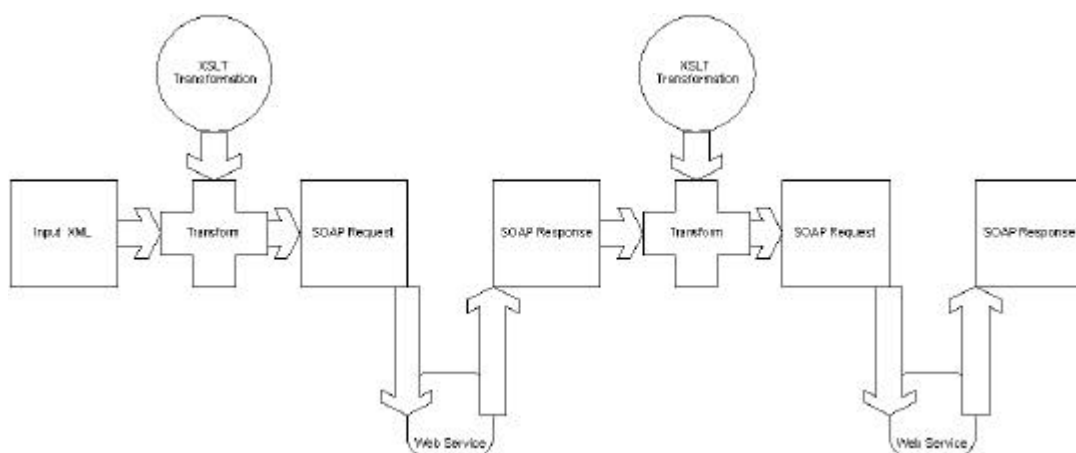
Simple Object Access Protocol (SOAP) as mentioned above is by far the most common data encoding that is used in Web Services. SOAP contains two parts encapsulated in an outer part. The outer part of SOAP is the envelope; it contains an optional header part, and a body part. The whole SOAP message is an XML document.

## Toolkits and Libraries

There are a large number of toolkits, libraries and APIs that make it easier to construct and use Web Services. Examples are .NET, IBM's Web Services Toolkit, Java XML Messaging API (JAXM), Apache SOAP etc. These toolkits perform a large number of things that make it easier for the developer to build Web Services and to interact with Web Services. They make it easier to take existing objects (COM, JavaBeans etc.) and expose their methods as Web Services. Most will create WSDL documents from Objects. Some will create client stub code from WSDL or deployed Web Services. Some provide interfaces into a UDDI repository. Remember however that to use a Web Service you only need to create a suitable XML document and process an XML response. Creating client stub code may initially help you perform this, but it also means that you will have to create code to call the client stub code.

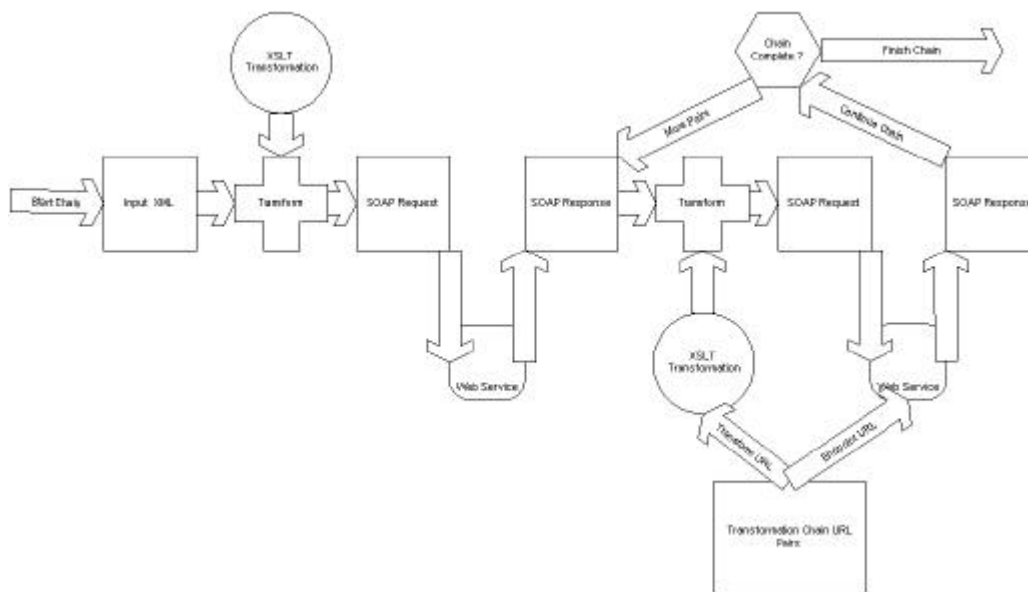
## Web Services Processes with Transformation

There is another way in which to interact with Web Services - by process and transformation. It is a relatively simple task to take an XML document and transform it into a SOAP XML document for delivery to a Web Service. The response of the first Web Service can then be transformed into another SOAP XML document for delivery to a second Web Service. In this example the transformation can be performed by XSLT, itself an XML document, and it is possible to create these transformation documents directly from the WSDL descriptor for the Web Service. In this way it is possible to integrate Web Services into an application without ever writing a piece of code.



Procession uses this technique to integrate a chain of Web Services into a single operation. The Procession process map instances a Web Services task. That task is specialised from the generic type by defining a source XML document and a transform that reformats the source XML document into a request for the

first Web Service. It then specifies a transformation to link the response from each Web Service into the request for the next Web Service, hence forming a sequence of Web Services that can be invoked. In this way the Procession process initiates a chain of Web Services.



The invocation of a chain may be performed as a synchronous operation, a task under direct control of the Procession process engine, or as is more often the case, delivered to the SSXMQ component to be executed as an asynchronous operation.

For the business people who know what their process is and understand how their business benefits from this process the impact is significant. It means that they can build Web Service chains and use them in their processes without having to refer to an internal IT department for specialist programmer support. No code is required to implement a Web Services chain.

Those of you who have been exposed to XML and transformations are starting to question this statement. The creation of XSLT transformations is not a trivial matter. However remember that the Web Services have been described by WSDL documents, which are themselves XML documents, and the XSLT Transformation is also an XML document. So Procession has a tool that creates the XSLT Transform from the WSDL document describing the Web Service and the Schema of the input XML. In the case of transformations later in the chain this is often the Schema defined by the previous Web Service. Hence no code needs to be written, and XSLT Transformations are derived from WSDL documents.

## Service Discovery

One of the strongest aspects of Web Services is the ability to discover a Web Service by asking for it by naming its capabilities. Once the Web Service is discovered its WSDL document is retrieved, the client code is created and the service is used. This approach may seem ideal, and for some types of Web Service there may be no problem. Discovering a stock price or the weather may be a good example of a Web Service which you discover automatically, but there are some Web Services which you would like to approve before transferring valuable information to them. Imagine a credit card handler Web Service or an International Money Order Web Service. I would like to "vet" such a service before using it. Standards like ebXML go some way to alleviating these kinds of issues by standardising the naming and specification of Web Services within UDDI, but human approval is still preferable to automatic integration.

Procession has integrated the UDDI aspect of Web Services into the deployment stage of building a Web Services chain. Hence when the service-to-service transformations are built, UDDI is used to discover the required Web Services. When this approach is used, the Process Designer only need specify the service requirements of each step in the chain to the Web Service Task and then the deployer will insert approved Web Services to deliver the requested services.

## Implementation Details

The approach to implementing the Web Services task type has been to focus on the generic framework rather than implementing specific adapters. We have not used a traditional Web Service Toolkit; it would not have given us full control over the transformation process. Instead we have used a standard transformation engine and coupled that with streaming protocol handlers. This enables the initiating task to define a sequence of URL's that define the starting point XML document, and a sequence of Web Service endpoints with transformations between the endpoints.

## Web Services Chain Implementation details

### Input Document

The starting document may be a simple Web Service that produces an XML document from a simple URL encoded request, or the document itself. The Web Services task takes the resulting document and the first transform to convert it into a request document for the first Web Service.

### Transitional Chain

The calling process specifies URL pairs to the Web Services Task that define the input transformation and the endpoint to which the result of that transformation is sent. The source document for the input transformation is the output document of the previous transformation.

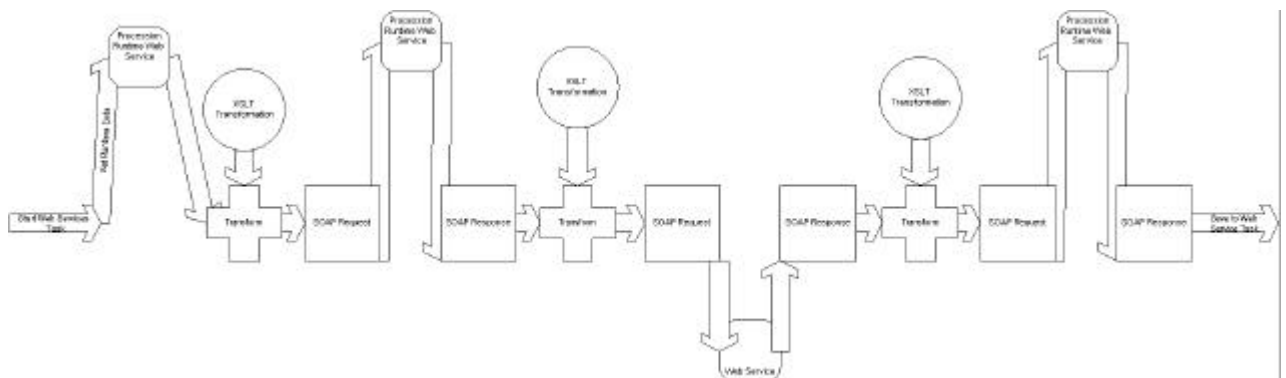
### Output Document

At the end of the chain there is an optional final transformation, and then the resulting document is placed in the Web Services Task output table, so that later tasks within the Procession process may access the result of the Web Service chain.

### Procession Integration

Since both the design time and run time contexts of the Procession Process Engine are exposed as Web Services, they may be used as part of a larger Web Service Chain. It is possible to access internal process data, as well as updating internal process data.

In practice a Web Service chain may initialise itself by invoking a process runtime Web Service. It would then invoke a number of external Web Services and finally reconnect to the process runtime via another process runtime Web Service. The final output stage might only transform a simple XML document to indicate success or a fault condition.



## Pluggable Protocols

As part of the implementation of the Web Service Task we have adopted standard protocol handling for both the Web Services Endpoint and the transformation document. Hence any location that can be expressed as a URL can be used, provided that that URL has a content handler registered with the Java virtual machine. Typically the http, https, ftp protocols are registered by default.

## Conclusion

The creation of Web Services has unified the software industry and will enable visionary CIOs to be completely vendor and technology agnostic and choose the best solution for their problem. Procession's framework approach to creating Web Service chains that integrate with human business processes eliminates future implementation risks within an integrated information architecture.