

What is SOAP ?

The Simple Object Access Protocol *SOAP* is a cross-platform and language-independent RPC protocol based on XML and, usually *butnotnecessarily* HTTP.

It uses XML to encode the information that makes the remote procedure call, and HTTP to transport that information across a network from clients to servers and vice versa.

SOAP has several advantages over other technologies like COM, CORBA etc: for example, its relatively cheap deployment and debugging costs, its extensibility and ease-of-use, and the existence of several implementations for different languages and platforms.

Please refer to out simple tutorial to understand SOAP.

This tutorial will make you familiar to the SOAP implementation for Ruby *SOAP4R*. This is a basic tutorial, so if you need a deep detail, you would need to refer other resources.

Installing SOAP4R:

SOAP4R is the SOAP implementation for Ruby developed by Hiroshi Nakamura and can be downloaded from:

NOTE: There may be a great chance that you already have installed this component.

```
Download SOAP
```

If you are aware of **gem** utility then you can use following command to install SOAP4R and related packages.

```
$ gem install soap4r --include-dependencies
```

If you are working on Windows then you need to download a zipped file from the above location and need to install it using standard installation method by running *ruby install.rb*.

Writing SOAP4R Servers:

SOAP4R supports two different types of servers:

- CGI/FastCGI based *SOAP::RPC::CGIStub*
- Standalone *SOAP::RPC:StandaloneServer*

This tutorial will give deatail on writing a stand alone server. There are following steps involved in writing a SOAP server:

Step 1 - Inherit SOAP::RPC::StandaloneServer Class:

To implement your own stand alone server you need to write a new class which will be child of *SOAP::StandaloneServer* as follows:

```
class MyServer < SOAP::RPC::StandaloneServer
  ..............
end
```

NOTE: If you want to write a FastCGI based server then you need to take *SOAP::RPC::CGIStub* as parent class, rest of the procedure will remain same.

Step 2 - Define Handler Methods:

Second step is to write your Web Services methods, which you would like to expose to the outside world.

They can be written as simple Ruby methods. For example, let's write two methods to add two numbers and divide two numbers:

```
class MyServer < SOAP::RPC::StandaloneServer
   ..............

   # Handler methods
   def add(a, b)
      return a + b
   end
   def div(a, b)
      return a / b
   end
end
```

Step 3 - Expose Handler Methods:

Next step is to add our defined methods to our server. The *initialize* method is used to expose service methods with one of the two following methods:

```
class MyServer < SOAP::RPC::StandaloneServer
   def initialize(*args)
      add_method(receiver, methodName, *paramArg)
   end
end
```

Here is the description of the parameters:

Paramter	Description
receiver	The object that contains the methodName method. you define the service methods in the same class as the methodDef method, this parameter is *self*.
methodName	The name of the method that is called due to a RPC request.
paramArg	Specifies, when given, the parameter names and parameter modes.

To understand the usage of *inout* or *out* parameters, consider the following service method that takes two parameters *inParamandinoutParam*, returns one normal return value *retVal* and two further parameters: *inoutParam* and *outParam*:

```
def aMeth(inParam, inoutParam)
   retVal = inParam + inoutParam
   outParam = inParam . inoutParam
   inoutParam = inParam * inoutParam
   return retVal, inoutParam, outParam
end
```

Now, we can expose this method as follows:

```
add_method(self, 'aMeth', [
    %w(in inParam),
    %w(inout inoutParam),
    %w(out outParam),
    %w(retval return)
])
```

Step 4 - Start the Server:

The final step is to start your server by instantiating one instance of the derived class and calling

start method.

```
myServer = MyServer.new('ServerName',
                        'urn:ruby:ServiceName', hostname, port)

myServer.start
```

Here is the description of required parameters :

Paramter	Description
ServerName	A server name, you can give what you like most.
urn:ruby:ServiceName	Here *urn:ruby* is constant but you can give a unique *ServiceName* name for this server.
hostname	Specifies the hostname on which this server will listen.
port	An available port number to be used for the web service.

Example:

Now, using above steps, let us write one standalone server:

```
require "soap/rpc/standaloneserver"

begin
   class MyServer < SOAP::RPC::StandaloneServer

      # Expose our services
      def initialize(*args)
         add_method(self, 'add', 'a', 'b')
         add_method(self, 'div', 'a', 'b')
      end

      # Handler methods
      def add(a, b)
         return a + b
      end
      def div(a, b)
         return a / b
      end
  end
  server = MyServer.new("MyServer",
            'urn:ruby:calculation', 'localhost', 8080)
  trap('INT'){
     server.shutdown
  }
  server.start
rescue => err
  puts err.message
end
```

When executed, this server application starts a standalone SOAP server on *localhost* and listens for requests on *port* 8080. It exposes one service methods, *add* and *div*, which takes two parameters and return the result.

Now, you can run this server in background as follows:

```
$ ruby MyServer.rb&
```

Writing SOAP4R Clients:

The *SOAP::RPC::Driver* class provides support for writing SOAP client applications. This tutorial will

describe this class and demonstrate its usage on the basis of an application.

Following is the bare minimum information you would need to call a SOAP service:

- The URL of the SOAP service *SOAPEndpointURL*
- The namespace of the service methods *MethodNamespaceURI*
- The names of the service methods and their parameters

Now, we will write a SOAP client which would call service methods defined in above example, named *add* and *div*.

Here are the main steps to create a SOAP client:

Step 1 - Create a SOAP Driver Instance:

We create an instance of *SOAP::RPC::Driver* by calling its new method as follows:

```
SOAP::RPC::Driver.new(endPoint, nameSpace, soapAction)
```

Here is the description of required parameters :

Paramter	Description
endPoint	URL of the SOAP server to connect with.
nameSpace	The namespace to use for all RPCs done with this SOAP::RPC::Driver object.
soapAction	A value for the SOAPAction field of the HTTP header. If *nil* this defaults to the empty string ""

Step 2 - Add Service Methods:

To add a SOAP service method to a *SOAP::RPC::Driver* we can call the following method using *SOAP::RPC::Driver* instance:

```
driver.add_method(name, *paramArg)
```

Here is the description of the parameters:

Paramter	Description
name	The name of the remote web service method.
paramArg	Specifies the names of the remote procedures' parameters.

Step 3 - Invoke SOAP service:

The final step is to invoice SOAP service using *SOAP::RPC::Driver* instance as follows:

```
result = driver.serviceMethod(paramArg...)
```

Here *serviceMethod* is the actual web service method and *paramArg...* is the list parameters required to pass in the service method.

Example:

Based on the above steps, we will write a SOAP client as follows:

```ruby
#!/usr/bin/ruby -w

require 'soap/rpc/driver'

NAMESPACE = 'urn:ruby:calculation'
URL = 'http://localhost:8080/'

begin
   driver = SOAP::RPC::Driver.new(URL, NAMESPACE)

   # Add remote sevice methods
   driver.add_method('add', 'a', 'b')

   # Call remote service methods
   puts driver.add(20, 30)
rescue => err
   puts err.message
end
```

Further Readings:

I have explained you just very basic concepts of Web Services with Ruby. If you want to drill down it further, then there is following link to find more details on Web Services with Ruby.