# Lab 9: RPC and Database Connection with Ruby

Learning Objectives:

1. Introduction to Remote Procedure Call (RPC)
2. Using XML-RPC with Ruby
3. Access a Database using Ruby
4. References

Supporting files:

1. client.rb
2. server.rb
3. sqldemo.rb

---

## 1. Introduction to Remote Procedure Call (RPC)

Remote Procedure Call (RPC) is a protocol that provides the high-level communications paradigm used in the operating system. RPC presumes the existence of a low-level transport protocol, such as Transmission Control Protocol/Internet Protocol (TCP/IP) or User Datagram Protocol (UDP), for carrying the message data between communicating programs. RPC implements a logical client-to-server communications system designed specifically for the support of network applications.

The RPC protocol enables users to work with remote procedures as if the procedures were local. The remote procedure calls are defined through routines contained in the RPC protocol. Each call message is matched with a reply message. The RPC protocol is a message-passing protocol that implements other non-RPC protocols such as batching and broadcasting remote calls. The RPC protocol also supports callback procedures and the select subroutine on the server side.

A client is a computer or process that accesses the services or resources of another process or computer on the network. A server is a computer that provides services and resources, and that implements network services. Each network service is a collection of remote programs. A remote program implements remote procedures. The procedures, their parameters, and the results are all documented in the specific program's protocol.

RPC provides an authentication process that identifies the server and client to each other. RPC includes a slot for the authentication parameters on every remote procedure call so that the caller can identify itself to the server. The client package generates and returns authentication parameters. RPC supports various types of authentication such as the UNIX and Data Encryption Standard (DES) systems.

In RPC, each server supplies a program that is a set of remote service procedures. The combination of a host address, program number, and procedure number specifies one remote service procedure. In the RPC model, the client makes a procedure call to send a data packet to the server. When the packet arrives, the server calls a dispatch routine, performs whatever service is requested, and sends a reply back to the client. The procedure call then returns to the client.

---

## 2. Using XML-RPC with Ruby

XMLRPC is a lightweight protocol that enables remote procedure calls over HTTP. It is defined at www.xmlrpc.com. XMLRPC allows you to create simple distributed computing solutions that span computer languages. Its distinctive feature is its simplicity compared to other approaches like SOAP and CORBA.

The Ruby standard library package ‘xmlrpc’ enables you to create a server that implements remote procedures and a client that calls them. Very little code is required to achieve either of these

More information on XMLRPC can be found here.

**XMLRPC::Server**

Implements a standalone XML-RPC server. The serve() method is called after all the handlers are added to the server. This method starts the server to listen for XML-RPC requests and answer them.

More information on XMLRPC::Server can be found here.

An example of XMLRPC::Server is as follows:

```
require "xmlrpc/server"

port = 50500

#
# This year the ports 50500 to 50550 are open, its suggested to work only within ports 50500-50550
#

class Num
  INTERFACE = XMLRPC::interface("num") {
  meth 'int add(int, int)', 'Add two numbers', 'add'
  meth 'int div(int, int)', 'Divide two numbers'
}
  def add(a, b) a + b end
  def div(a, b) a / b end
end

server = XMLRPC::Server.new(port, ENV['HOSTNAME'])
server.add_handler(Num::INTERFACE,  Num.new)
server.serve
```

**XMLRPC::Client**

Provides remote procedure calls to a XML-RPC server. After setting the connection-parameters with ::new which creates a new XMLRPC::Client instance, you can execute a remote procedure by sending the #call or #call2 message to this new instance.

The given parameters indicate which method to call on the remote-side and of course the parameters for the remote procedure.

More information on XMLRPC::Client can be found here.

An example of XMLRPC::Client is as follows:

```
require "xmlrpc/client"

#
# This year the ports 50500 to 50550 are open, its suggested to work only within ports 50500-50550
#

ourServer = XMLRPC::Client.new(ENV['HOSTNAME'], "/RPC2", 50500)

# Example: to connect to e5-01-02, you need to type
#ourServer = XMLRPC::Client.new('129.128.211.42', "/RPC2", 50500)

puts "Add 5 and 3"
puts ourServer.call('num.add', 5, 3)

puts "Div 24 and 6"
puts ourServer.call('num.div', 24, 6)

aProxy = ourServer.proxy("num")

puts "Proxy add 6 and 10"
puts aProxy.add(6, 10)

puts "Proxy div 70 by 5"
puts aProxy.div(70, 5)
```

It is important to stick to the ports mentioned in the above examples. You could also replace the parameter ENV['HOSTNAME'] with 'localhost', if using the same computer as both server and client. The above examples are shared with you on eclass. It is recommended that you try and run them, play around with them, and ask any question you have.

---

## 3. Access a Database using Ruby

The Ruby DBI module provides a database-independent interface for Ruby scripts similar to that of the Perl DBI module. DBI stands for Database Independent Interface for Ruby, which means DBI provides an abstraction layer between the Ruby code and the underlying database, allowing you to switch database implementations really easily. It defines a set of methods, variables, and conventions that provide a consistent database interface, independent of the actual database being used. DBI can interface with databases such as, MySQL, ODBC, Oracle, OCI8 (Oracle), Proxy/Server, SQLite etc.

In this lab, we will discuss how to connect to MySQL with Ruby. The following steps show you how to install and connect to MySQL with Ruby. At first, in order to install MySQL gem, in your terminal you need to type:

```
$ gem install mysql
```

Then in order to start your MySQL server, you can type this at the terminal:

```
$ mysqld
```

Now, open a new command window run the mysql client program from the command line, as:

```
$ mysql --user=root mysql
```

Next, create a database named ‘ece421grp1’ in Ruby as follows:

```
mysql> create database ece421grp1;
create table animal(name VARCHAR(40), category VARCHAR(40));
```

Populate the above table with some data and as a first exercise we try to connect to the MySQL server and print all the names in the table animal.

```
require 'mysql'

#my = Mysql.new(hostname, username, password, databasename)
con = Mysql.new('localhost', '', '', ‘ece421grp1’)
rs = con.query('select * from animal)
rs.each_hash { |h| puts h['name']}
con.close
```

An example of connecting to a MySQL database with ruby is attached as sqldemo.rb. Here you can have a detailed look at the API.

## 4. References:

[1] https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/com.ibm.aix.progcomc/ch8_rpc.htm
[2] https://ruby-doc.org/stdlib-2.3.1/libdoc/xmlrpc/rdoc/XMLRPC.html
[3] https://ruby-doc.org/stdlib-2.3.1/libdoc/xmlrpc/rdoc/XMLRPC/Server.html
[4] https://ruby-doc.org/stdlib-2.3.1/libdoc/xmlrpc/rdoc/XMLRPC/Client.html
[5] http://rubylearning.com/satishtalim/ruby_mysql_tutorial.html