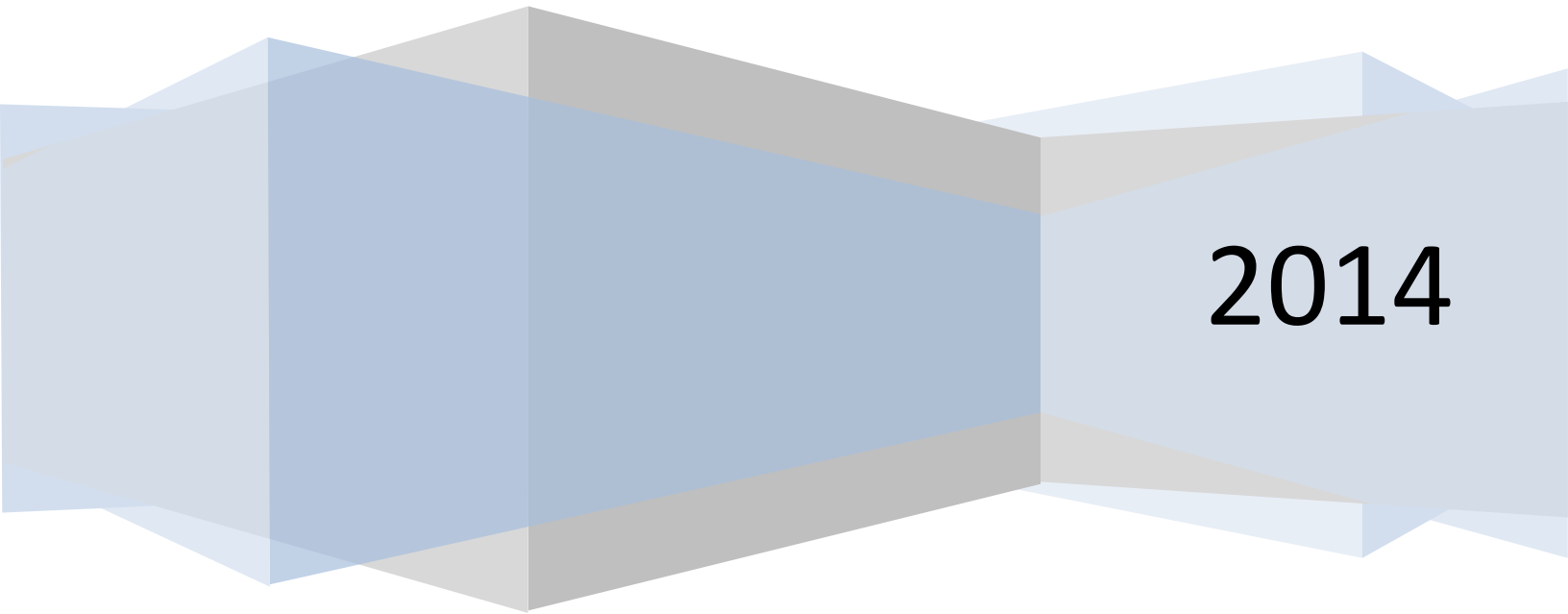
Conestoga College - SET

Assignment #1

Developing *services* and *consumers* in a SOA



2014

Contents

Introduction - Putting it all Together …

This assignment will challenge you and your team (maximum 4 people – recommended in your *CAPSTONE* groups) to use and confront all of the design elements and development concerns of a Service-Oriented Architecture (SOA). This assignment will require you to develop both services to be *published* in the SOA and user applications to *consume* the services.

In the end, you will demonstrate your working services and user application in the lab within the SOA – along with every other team's services and applications running in tandem.

In A *Nut Shell*

We're going to work within the confines of a SOA developed by **Omnicorp Corporation**[1]. Thorton and his team have put a lot of effort into developing this SOA schema. Their messaging scheme is setup and configured along the lines of the medical industry's HL7 v2.x schema.

The objective of this assignment is for you and your team to develop and publish a number of required services and also develop another application that you can use to call upon the other services published in the SOA to perform some sort of useful functionality. So in essence, you will be the creators and publishers of services within the SOA and you will also be the consumers of services published in the SOA.

As you begin to use and understand the SOA's schema, you will appreciate just how easy it can be to construct a portion (or all) of a business' work-flow using a SOA. You can quickly develop and publish new services for use and as well, you can quickly develop and put together applications to consume the services and call them in whatever order you may need to.

Thorton and his team need you to develop a number of services designed to support GIORP-5000 product line from TPS-III Industries, the EMS Solution from Omnicorp Corporation as well as various other supporting services. Also as mentioned, he needs you to create an application used to demonstrate the calling of these services. The services he has in mind are :

- **Purchase-Totaller** for the GIORP-5000
  - Given a customer's total GIORP purchase as well as the region they live in in Canada, you will develop a service to determine and return the sub-total, provincial sales tax – PST (if any), harmonized sales tax – HST (if any) and the goods and services tax – GST (if any) as well as the purchase grand total.
- **Pay-Stub Amount Generator** for the EMS System
  - Given an employee's type, their hours worked, their salary, etc. you will develop a service to return the amount of money they receive on their pay cheque.
- **Car-Loan Calculator**
  - Thorton is thinking of getting into the Auto industry and he would like you to develop a service that given the purchase amount of a car and the going loan interest rate determines and returns loan payment amounts for various loan terms (36 months, 48 months and 60 months)
- **Canadian Postal-Code Validator**
  - Thorton is helping Canada Post by developing this service for them (actually you are developing it and he will be selling it to them ☺). Given a mailing address' province and suggested postal code, this service determines if the postal code is valid.

[1] This highly successful company is owned and operated by our friend, Thorton P. Snodgrass III

Assignment Requirements

First let's start by discussing the design considerations, implementation languages and destination OS's ...

Communication, Development Languages and OS's

The underlying communication scheme that you will use used to support the SOA Registry and SOA messaging protocol is the stream ***socket.***

In order to demonstrate the flexibility of a SOA, you are required to develop the 4 services in different development languages and running on different OS's as follows :

- **Purchase-Totaller** for the GIORP-5000
  - Developed in C# running in Windows 7 OS
  - See this link for ideas and help on using sockets in C#
- **Pay-Stub Amount Generator** for the EMS System
  - Developed in C running in Windows 7 OS
  - See this link for ideas and help on using sockets in C
- **Car-Loan Calculator**
  - Developed in Java running in Windows 7 OS
  - See this link for ideas and help on using sockets in Java
- **Canadian Postal-Code Validator**
  - Developed in C++ running in Windows 7 OS
  - See this link for ideas and help on using sockets in C++

NOTE : Many of the links above take you to www.codeproject.com and have source code associated with the notes found there. In order to download the source code you will need to create and use a CodeProject account – it is well worth it to do this …

Feel free to use this source code as a basis to your solution …

Refer to the section on **Service Logic** (starting on Page 5 of this assignment) for the internal logic of each of these services.

I have developed and provided the SOA Registry for you to use. The runtime (and necessary supporting files / directory structure) are found in this ZIP file associated with this assignment. The SOA Registry was written in Java and requires you to customize a few files before trying it out in your development – **please see the README file in the extracted "SOA" root directory**.

Service Design and Runtime Logging

When designing your different SOA services, it is important that you follow best practices and ensure that your service(s) are and have :

- Modular design – to maximize re-use of code and cohesive design
- Full class/file, method/function and inline commenting
- Full error handling / input validation capabilities when talking to the SOA Registry and other calling applications
    - You will need to consider the purpose of the service and what type of error handling and validation you need to provide
- Retrieve their runtime settings from a configuration file at start-up
    - Much in the same way as the SOA Registry from was given to you
    - This will allow you to be able to develop your SOA more easily and be able to more fully test it!

It is also important that each of your services log information during their execution. Your team will be judged and marked based upon your source code as well as the various run-time logs produced by your services on demo day. Please ensure that all log entries are date and time stamped and that the following events and information is logged :

- Your team name as well as the service's tagName and serviceName
- Messages going out to SOA Registry as well as the responses that come back
- Incoming service requests (Execute Service message) contents including
    - The calling team and their ID
    - The serviceName, arguments and values
- The response that your service creates to send back to the requesting team
- Any and all error conditions and exception conditions (including socket errors)
- See example Service Log in Appendix A of this assignment

Keep in mind that the team name you choose for your 4 services must all be the same. As well this team name should be branded on your consumer application….

Consumer Design and Runtime Logging

You also need to develop a Windows GUI based program (in C#) in order to demonstrate the calling of the SOA Registry and other services. This application needs to be able to prompt the user for required information as necessary. For example if the user decides to call the CAR-LOAN service, your SOA user application must prompt the user for the arguments as specified in the SOA Registry's response to the Query Service Message. As an added feature, the user application can also do some input validation on these argument values given that you were told the data-type of the arguments as well.

From the user perspective, your client application should work is as follows :

- The first screen should ask the user which of the 4 services they wish to execute – and the user can select
    - When a selection is made, your application puts out a call to the SOA-Registry looking for a service to perform that task
- The next screen is presented to the user after your application has asked the SOA-Registry for information about the selected service. This is the screen where your client application needs to parse and understand the information coming back about the service to call in terms of the service's name, parameter names and data-types and optionality.
    - This screen should display the team name that has published the service about to be executed, the service name and the service description
    - This screen will allow the user to enter their input information and select an EXECUTE function

- Once the user has EXECUTED, your client application may wish to run simple data-type validation on the user inputs before calling the actual service. At some point, your application calls the service that you were given with the user's input information and awaits a result / response
    - When the client application receives the response, this input screen will be updated with the service's response / results
    - The user will be able to select a DONE option which takes them back to the first screen of your application once again

Just as in the case of the individual services, the user application must produce a log file as well – it is part of the mark you will receive on the application. See the example User-App Log in Appendix B of this assignment.

Other Considerations

Given that you are developing 4 different services in 4 different languages, you might actually want to use some of your PAD knowledge to define a generic socket-communication interface with OS and language specific code behind it. This is an example of an *Abstraction Layer* and as far as having a SOA is concerned, this type of design abstraction is the basis of SOA Interface policies.

As well, keep in mind that the SOA Registry can be configured to assign time-outs to your team and service registration. So it may be possible that the Registry will drop your team and service information at some point. You will need to develop a strategy using the existing SOA Registry commands to detect this situation and properly react to it.

When is this thing due ?

As mentioned in class, this assignment is officially due Sunday November 30, 2014 by 11:59pm in the eConestoga Dropbox. You will need to submit the source code, any makefiles – or build scripts that are needed to recompile and regenerate your executable(s) as well as any configuration files that your services may need.

Also as mentioned in class, you and your team will end up demoing your services and client on December 1, 2014 during the last SOA class period of the term – at the same time as every other team is demoing … so it might not be a bad idea with make sure that your team's services can co-exist and *play nicely* with other services from other teams …

Service Logic

Purchase-Totaller for the GIORP-5000

- The tagName for this service must be GIORP-TOTAL, you choose your service's security level from a value of 1 (LOW), 2 or 3 (HIGH).
- You may name your specific service however you like (e.g. giorpTotaller or totalPurchase)
    - The service needs to take 2 mandatory arguments – these arguments can have any name or argument order but the argument values need to be passed into this service
        - One argument needs to capture the Province or Territory that the purchase is being made in – allowable argument values are shown in the table below
        - The other argument needs to be the total value (in dollars and cents) of the purchase being made
    - You may add other arguments if you like
- The idea behind this service is to calculate and return the purchase sub-total, various tax additions and grand-total for the purchase given the Province/Territory it is being made in – the following table shows the Provincial / HST and GST rates for the various regions
- In regions where an HST rate is specified – there is no additional PST and GST calculated (and vice versa)

Allowed Region Code	Code Meaning	PST Rate	HST Rate	GST Rate
NL	Newfoundland	--	13%	--
NS	Nova Scotia	--	15%	--
NB	New Brunswick	--	13%	--
PE	Prince Edward Island	10% *		5%
QC	Quebec	9.5% **		5%
ON	Ontario	--	13%	--
MB	Manitoba	7%		5%
SK	Saskatchewan	5%		5%
AB	Alberta	0%		5%
BC	British Columbia	--	12%	--
YT	Yukon Territories	0%		5%
NT	Northwest Territories	0%		5%
NU	Nunavut	0%		5%

* In PEI, the PST is calculated on the purchase price + GST. eg. If you purchase something for $100.00, then the GST is $5.00 and the PST is $10.50 (=10% of $105.00)

** In Quebec, the PST is calculated on the purchase price + GST. e.g. if you purchase something for $100.00, then the GST is $5.00 and the PST is $9.98 (=9.5% of $105.00)

- This service must return 5 different values – the names of the return values can be whatever you like and specified in any order, but must capture the following information
    - Sub-total amount – which is the value that was passed into the service as the purchase amount
    - PST amount – the value to be added to the purchase amount given the information in the above table

- HST amount – the value to be added to the purchase amount given the information in the above table
- GST amount – the value to be added to the purchase amount given the information in the above table
- Total purchase amount – the sub-total amount plus the PST amount plus the HST amount plus the GST amount

Pay-Stub Amount Generator for the EMS System

- The tagName for this service must be PAYROLL, you choose your service's security level from a value of 1 (LOW), 2 or 3 (HIGH).
- You may name your specific service however you like (e.g. emsPayStub or payCheckMaker)
  - The service needs to take 3 mandatory arguments – these arguments can have any name or argument order but the argument values need to be passed into this service
    - One argument needs to capture the employee type – allowable argument values are shown in the table below
    - Another argument needs to capture the total hours worked (e.g. 44.5)
    - Another argument needs to capture the employee's rate of pay
  - This service also takes 2 optional arguments
    - One is for SEASONAL employees – this argument represents the number of pieces made
    - The other is for CONTRACT employees and represents the number of weeks that the contract covers
  - You may add other arguments if you like
- The idea behind this service is to take the employee type and calculate the amount of pay that will show up on the employee's pay cheque
  - Depending on the employee type, the total-pay amount is calculated differently
  - The different formulae are specified in the table below – where
    - H represents the hours worked
    - S represents the rate-of-pay
    - P represents the pieces made (SEASONAL only)
    - W represents the duration (in weeks) of the contract

Allowed Employee Type	Code Meaning	Meaning of S	Formula
HOUR	Part-Time Employee	Hourly rate (e.g. 15.23)	• if H<=40.0 then Pay=(H * S) • if H>40.0 then Pay=(40.0 * S) + [(H – 40.0) * (S * 1.50)]
FULL	Full-Time Employee	Yearly Salary (e.g. 56000.00)	• Pay=S/52
SEASON	Seasonal Employee	Piece Pay (e.g. 2.30)	• if H<40.0 then Pay=(P * S) • if H>=40.0 then Pay = (P * S) + 150.00
CONTRACT	Contract Employee	Total Contract Value (e.g. 250000.00)	• regardless of H, Pay=(S / W)

- This service needs to only return one value – the name of the return value can be whatever you like, but must capture the following information
  - Total Pay Value – which is the amount to show on the employee's weekly pay cheque

Car-Loan Calculator

- The tagName for this service must be CAR-LOAN, you choose your service's security level from a value of 1 (LOW), 2 or 3 (HIGH).
- You may name your specific service however you like (e.g. carPayments or loanMaker)
    - The service needs to take 2 mandatory arguments – these arguments can have any name or argument order but the argument values need to be passed into this service
        - One argument needs to capture the principal amount of the loan (i.e. the purchase price of the car)
        - Another argument needs to capture the interest rate on the loan (specified as a value like 4.25)
    - You may add other arguments if you like
- The idea behind this service is to take the purchase price for a car and determine the total loan value (including compound interest) in order to return the monthly loan payment for a 36, 48 and 60 month term
    - Check out this reference for the formula (including example) on how to calculate a loan payment given an annual interest rate, the principal amount and the term of the loan
- This service needs to return 3 values – the names of the return values can be whatever you like and specified in any order, but must capture the following information
    - 36 Month Payment Amount – which is the amount of the car loan if paid over 3 years
    - 48 Month Payment Amount – which is the amount of the car loan if paid over 4 years
    - 60 Month Payment Amount – which is the amount of the car loan if paid over 5 years

Canadian Postal-Code Validator

- The tagName for this service must be POSTAL, you choose your service's security level from a value of 1 (LOW), 2 or 3 (HIGH).
- You may name your specific service however you like (e.g. goingPostal or checkPostalCode)
    - The service needs to take 2 mandatory arguments – these arguments can have any name or argument order but the argument values need to be passed into this service
        - One argument needs to capture the Province or Territory that the postal code is from – allowable argument values are shown in the table below
        - The other argument needs to be the postal code that you are validating (e.g. N2G4E4, N2G 4E4, n2g 4e4, etc.)
    - You may add other arguments if you like
- The idea behind this service is that given the province / territory in Canada and a person's postal code you can somewhat validate that it is correct and where possible also return special notes about the region given the postal code.
    - For a complete set of areas, allowed rules, etc. check this out – NOTE: You don't need to implement the complete set of rules – only those specified in the tables below …

Allowed Region Code	Code Meaning	Postal Code must begin with	Region's Special Notes
NL	Newfoundland	• A0, A1, A2, A5 or A8	• A0 = Rural Newfoundland • A3, A4, A6, A7 or A9 = Wanna Be Newfoundland
NS	Nova Scotia	• B0, B1, B2, B3, B4, B5, B6 or B9	• B0 = Rural Nova Scotia • B7 or B8 = Wanna Be Nova Scotia
NB	New Brunswick	• E1, E2, E3, E4, E5, E6, E7, E8 or E9	• E0 = Wanna be Rural New Brunswick
PE	Prince Edward Island	• C0 or C1	• C0 = Rural PEI • C2, C3, C4, C5, C6, C7, C8 or C9 = Wanna be PEI
QC	Quebec	• G0, G1, G2, G3, G4, G5, G6, G7, G8 or G9 • H0, H1, H2, H3, H4, H5, H7, H8 or H9 • J0, J1, J2, J3, J4, J5, J6, J7, J8 or J9	• G0, H0, J0 = Rural Quebec • G1A = Provincial Government • H0H 0H0 = Santa Claus • G = Eastern Quebec • H = Montreal Area • J = Western / Northern Quebec • H6 = Wanna be Quebec
ON	Ontario	• K0, K1, K2, K4, K6, K7, K8 or K9 • L0, L1, L2, L3, L4, L5, L6, L7, L8 or L9 • M1, M2, M3, M5, M6, M7, M8 or M9 • N0, N1, N2, N3, N4, N5, N6, N7, N8 or N9 • P0, P1, P2, P3, P4, P5, P6, P7, P8 or P9	• K0, L0, N0, P0 = Rural Ontario • K1A = Government of Canada • M7A = Queen's Park • K = Eastern Ontario • L = Central Ontario • M = Toronto Area • N = Western Ontario • P = Northern Ontario • K3, K5, M0, M4 or M6 = Wanna be Ontario
MB	Manitoba	• R0, R1, R2, R3, R4, R5, R6, R7, R8 or R9	• R0 = Rural Manitoba • R2 or R3 = Winnipeg Area
SK	Saskatchewan	• S0, S2, S3, S4, S6, S7 or S9	• S0 = Rural Saskatchewan • S7 = Saskatoon Area • S1, S5 or S8 = Wanna be Saskatchewan

Allowed Region Code	Code Meaning	Postal Code must begin with	Region's Special Notes
AB	Alberta	• T0, T1, T2, T3, T4, T5, T6, T7, T8 or T9	• T0 = Rural Alberta • T5 or T6 = Edmonton Area • T2 or T3 = Calgary Area
BC	British Columbia	• V0, V1, V2, V3, V4, V5, V6, V7, V8 or V9	• V0 = Rural British Columbia
YT	Yukon Territories	• Y0A, Y0B or Y1A	• Y0 = Remote Yukon Areas • Y1A = Whitehorse • Y2, Y3, Y4, Y5, Y6, Y7, Y8 or Y9 – Wanna be Yukon
NT	Northwest Territories	• X0E, X0F or X1A	• X0 = Remote NWT Areas • X1A = Yellowknife • X2, X3, X4, X5, X6, X7, X8 or X9 – Wanna be NWT
NU	Nunavut	• X0A, X0B or X0C	• Everything is remote in Nunavut

NOTE: Overriding Rules

- This rule applies to postal codes from all regions – the letters D, F, I, O, Q and U cannot appear *anywhere* in the postal code
- The postal code format must be ANANAN – where A represents a valid alpha character and N represents a valid numeric character
- If either of these rules are broken, then you need to return a NOT-VALID state and also special notes indicating why

- This service must return 2 different values – the names of the return values can be whatever you like and specified in any order, but must capture the following information
  - Postal-Code Valid – returns a VALID or NOT-VALID depending on what you find
    - Note that the not valid postal codes are made up of anything not found in the "Postal Code must begin with" column or marked as a "Wanna be" in the special notes
  - Special Notes – the set of special notes given to the postal code

SOA Messaging Protocol and Scheme

SOA Messaging Scheme

As you know from Lesson 1, a SOA is really based upon a clear, well defined set of rules, messaging schemes and protocols. At the heart of any SOA is the Registry where information about these policies, the services, etc. is stored and enforced. I have developed the SOA-Registry that you must use – it adheres to the schema and protocols discussed in this assignment. (You can download it from the course Assignment space)

As mentioned, our SOA is based upon the HL7 v2.x messaging schema (please see page 12 of this assignment for a detailed description of the messaging protocol.

Below you will find a summary of the 6 different SOA processing messages and the 2 variations of allowed responses to each of the messages. Elements notes in ***bold italics*** are pieces of information that you need to provide when sending your messages. Examples of these messages are included in the sample client directory of the given SOA Registry zip file.

1. *Register Team* Message

Description : Before you can begin to do anything with the SOA Registry or its services, you need to register your team. This is done by sending a REG-TEAM message to the registry with your team name. If the registration succeeds, you will receive a ***teamID*** from the Registry and an expiration time as well (for your Registry license). The registry will also assign your team a security level (which it doesn't tell you about) which may prevent you from executing some services … ☺

Message Format :

```
DRC|REG-TEAM|||
INF|<team name>|||
```

Response Format : The SOA Registry will respond to your message with either an OK or NOT-OK status with the information specified below.

```
PASS:    SOA|OK|<teamID>|<expiration>||
FAIL:    SOA|NOT-OK|errorCode|errorMessage||
```

2. *Unregister Team* Message

Description : If for some reason you want to remove your team (and its services) from being published in the Registry – then you send an UNREG-TEAM message.

Message Format :

```
DRC|UNREG-TEAM|<team name>|<teamID>|
```

Response Format : The SOA Registry will respond to your message with either an OK or NOT-OK status with the information specified below.

```
PASS:    SOA|OK||||
FAIL:    SOA|NOT-OK|errorCode|errorMessage||
```

3. *Query Team* Message

Description : When a service you published receives a request to perform a task (i.e. your service receives an *Execute Service Message*), you need to verify that the calling team is allowed to perform that service. The response from this message indicates if the calling team is still valid (and hasn't expired) and if they have the necessary security level to call your service.

Message Format :

```
DRC|QUERY-TEAM|<team name>|<teamID>|
INF|<team name>|<teamID>|<service tag name>|
```

NOTE: The ***<team name>*** and ***<teamID>*** in the DRC segment represent your team's name and ID. The ***<team name>*** and ***<teamID>*** in the INF segment represent the team that you are asking about. The ***<service tag name>*** in the INF segment is the tagName of **your service** that this team wishes to execute.

Response Format : The SOA Registry will respond to your message with either an OK or NOT-OK status with the information specified below.

```
PASS:    SOA|OK||||
FAIL:    SOA|NOT-OK|errorCode|errorMessage||
```

4. *Publish Service* Message

Description : Call the SOA Registry with this message in order to publish a service for your team so that other teams may begin to call it. See commented example message for a description of this message's segments and elements.

Message Format :

```
DRC|PUB-SERVICE|<team name>|<teamID>|
SRV|<tag name>|<service name>|<security level>|<num args>|<num responses>|<description>|
ARG|<arg position>|<arg name>|<arg data type>|[mandatory | optional]||
      . . .
RSP|<resp position>|<resp name>|<resp data type>||
      . . .
MCH|<published server IP>|<published port>|
```

Response Format : The SOA Registry will respond to your message with either an OK or NOT-OK status with the information specified below.

```
PASS:    SOA|OK||||
FAIL:    SOA|NOT-OK|errorCode|errorMessage||
```

NOTE :

1. Only (somewhat) primitive data-types are allowed in the ARG and RSP segments. The allowed data-types are:
   - `char`
   - `short`
   - `int`
   - `long`
   - `float`
   - `double`
   - `string`
2. Select port to publish in a range greater than 2000.

5. *Query Service* Message

Description : If you need to call a service to perform some work for you – you send the SOA Registry this message asking for information about a published service capable of doing that work for you.

Message Format :

```
DRC|QUERY-SERVICE|<team name>|<teamID>|
SRV|<tag name>||||||
```

Response Format : The SOA Registry will respond to your message with either an OK or NOT-OK status with the information specified below. In the case of an OK response, you will receive the information about the service that you are allowed to call – so you will need to parse this response and use the information in it to construct your *Execute Service Message*.

```
PASS:    SOA|OK|||<num segments>|
         SRV|<team name>|<service name>||<num args>|<num responses>|<description>|
         ARG|<arg position>|<arg name>|<arg data type>|[mandatory | optional]||
               . . .
         RSP|<resp position>|<resp name>|<resp data type>||
               . . .
         MCH|<published server IP>|<published port>|

FAIL:    SOA|NOT-OK|errorCode|errorMessage||
```

NOTE :

1. The **<num segments>** value represents the number of message segments that follow the SOA|OK
2. Please note that the second field in the SRV tag of the SOA|OK response contains the team name of the team that published the service.

6. *Execute Service* Message

Description : After calling the SOA Registry and sending a *Query Service Message*, you need to parse out the response (assuming it was successful) and construct the following message which you send to the service's machine on its designated port. Remember that when your service receives this message from another team – you need to call the SOA Registry with the *Query Team Message* in order to see if that team is allowed to call your service or not.

Message Format :

```
DRC|EXEC-SERVICE|<team name>|<teamID>|
SRV||<service name>||<num args>|||
ARG|<arg position>|<arg name>|<arg data type>||<arg value>|
      . . .
```

Response Format : All of your published services need to adhere to the SOA messaging scheme and respond to the calling team with the following messages

```
PASS:    PUB|OK|||<num segments>|
         RSP|<resp position>|<resp name>|<resp data type>|<resp value>|
               . . .

FAIL:    PUB|NOT-OK|errorCode|errorMessage||
```

SOA Messaging Protocol

The HL7 v2.x messaging scheme uses sockets to communicate between different end points (different computer systems) within a medical facility. The HL7 scheme is based upon the idea of message passing – where each message as a single purpose. In this scheme, a message is comprised of multiple segments. Each segment within the message gives information about the purpose of the message. Each segment is comprised of multiple fields – and the first field in any segment indicates the purpose of that segment. A general HL7 v2.x message looks like this :

```
<BOM>[segment]<EOS>[segment]<EOS> . . . [segment]<EOS><EOM>
```

where

<BOM> is a special character used to mark and indicate the start of a new message
<EOS> is a special character used to mark and indicate the end of a message segment
<EOM> is a special character used to mark and indicate the end of a message

The placement and existence of the special message framing characters is the responsibility of the vendor using / implementing the schema (in this case – you and your team).

In our SOA messaging scheme the following ASCII character codes will be used for framing the messages :

<BOM> is represented by ASCII 11
<EOS> is represented by ASCII 13
<EOM> is represented by ASCII 28

So when you are building a message to send to either the SOA-Registry or a published service – you will need to construct the message and frame it with these special characters. One very special note worth mentioning is that in practice, the <EOM> framing character is followed by a carriage return on the socket. The SOA-Registry ensures that the <EOM> is followed by a carriage return.

If we re-examine one of the example Register Team messages (for team Blotto) as a stream of characters coming across a socket, you will see that the message :

```
DRC|REG-TEAM|||
INF|Blotto|||
```

is really transmitted as :

```
[11][68][82][67][124][82][69][71][45][84][69][65][77][124][124][124][13][73][78][70][124][66][108][111][116][116][111][124][124][124][13][28][13]
     D   R   C    |   R   E   G   -   T   E   A   M    |    |    |      I   N   F    |   B   l   o   t   t   o    |    |    |
```

where [###] represents the ASCII character codes … Notice the characters highlighted in yellow … does their placement and ASCII character code purpose give you any ideas about how to communicate (read and write) with the socket ?? The character highlighted in green marks the <BOM> and the character highlighted in red marks the <EOM>

Appendix A - The Service Run-Time Log

```
2012-09-03 17:30:00 ======================================================
2012-09-03 17:30:00 Team    : Blotto (Dan K, Larry D)
2012-09-03 17:30:01 Tag-Name: CAR-LOAN
2012-09-03 17:30:01 Service : myCarLoanCalculator
2012-09-03 17:30:02 ======================================================
2012-09-03 17:30:05 ---
2012-09-03 17:30:05 Calling SOA-Registry with message :
2012-09-03 17:30:05   >> DRC|REG-TEAM|||
2012-09-03 17:30:05   >> INF|Blotto|||
2012-09-03 17:30:08   >> Response from SOA-Registry :
2012-09-03 17:30:08      >> SOA|OK|1180|18:00:05||
2012-09-03 17:30:12 ---
2012-09-03 17:30:12 Calling SOA-Registry with message :
2012-09-03 17:30:12   >> DRC|PUB-SERVICE|Blotto|1180|
2012-09-03 17:30:12   >> SRV|CAR-LOAN|myCarLoanCalculator|3|2|3|Service to calculate loan payments|
2012-09-03 17:30:12   >> ARG|1|principalCarAmount|double|mandatory||
2012-09-03 17:30:12   >> ARG|2|interestRate|double|mandatory||
2012-09-03 17:30:12   >> RSP|1|payment36Month|double||
2012-09-03 17:30:12   >> RSP|2|payment48Month|double||
2012-09-03 17:30:12   >> RSP|3|payment60Month|double||
2012-09-03 17:30:12   >> MCH|142.112.50.103|50002|
2012-09-03 17:30:14   >> Response from SOA-Registry :
2012-09-03 17:30:14      >> SOA|OK||||
2012-09-03 17:30:30 ---
2012-09-03 17:30:30 Receiving service request :
2012-09-03 17:30:30   >> DRC|EXEC-SERVICE|Shcarpo|1183|
2012-09-03 17:30:30   >> SRV||myCarLoanCalculator||2|||
2012-09-03 17:30:30   >> ARG|1|principalCarAmount|double||15230.56|
2012-09-03 17:30:30   >> ARG|2|interestRate|double||3.75|
2012-09-03 17:30:05 ---
2012-09-03 17:30:35 Calling SOA-Registry with message :
2012-09-03 17:30:35   >> DRC|QUERY-TEAM|Blotto|1180|
2012-09-03 17:30:35   >> INF|Shcarpo|1183|CAR-LOAN|
2012-09-03 17:30:35   >> Response from SOA-Registry :
2012-09-03 17:30:35      >> SOA|NOT-OK|-5|Insufficient Security Level||
2012-09-03 17:30:36 ---
2012-09-03 17:30:36 Responding to service request :
2012-09-03 17:30:36      >> PUB|NOT-OK|-1|Sorry – your team has insufficient permissions to run this service||
```

Appendix B - The Consumer Run-Time Log

```
2012-09-03 17:30:00 ======================================================
2012-09-03 17:30:00              -- USER APP LOG --
2012-09-03 17:30:00 Team    : Shcarpo (Sid D, Larry S)
2012-09-03 17:30:02 ======================================================
2012-09-03 17:30:05 ---
2012-09-03 17:30:05 Calling SOA-Registry with message :
2012-09-03 17:30:05   >> DRC|REG-TEAM|||
2012-09-03 17:30:05   >> INF|Shcarpo|||
2012-09-03 17:30:08   >> Response from SOA-Registry :
2012-09-03 17:30:08      >> SOA|OK|1183|18:00:05||
2012-09-03 17:30:12 ---
2012-09-03 17:30:12 Calling SOA-Registry with message :
2012-09-03 17:30:12   >> DRC|QUERY-SERVICE|Shcarpo|1183|
2012-09-03 17:30:12   >> SRV|CAR-LOAN|||||
2012-09-03 17:30:14   >> Response from SOA-Registry :
2012-09-03 17:30:14      >> SOA|OK|||7|
2012-09-03 17:30:14      >> SRV|Blotto|myCarLoanCalculator||2|3| Service to calculate loan payments|
2012-09-03 17:30:14      >> ARG|1|principalCarAmount|double|mandatory||
2012-09-03 17:30:14      >> ARG|2|interestRate|double|mandatory||
2012-09-03 17:30:14      >> RSP|1|payment36Month|double||
2012-09-03 17:30:14      >> RSP|2|payment48Month|double||
2012-09-03 17:30:14      >> RSP|3|payment60Month|double||
2012-09-03 17:30:14      >> MCH|142.112.50.103|50002|
2012-09-03 17:30:30 ---
2012-09-03 17:30:30 Sending service request to IP 142.112.50.103, PORT 50002 :
2012-09-03 17:30:30   >> DRC|EXEC-SERVICE|Shcarpo|1183|
2012-09-03 17:30:30   >> SRV||myCarLoanCalculator||2|||
2012-09-03 17:30:30   >> ARG|1|principalCarAmount|double||15230.56|
2012-09-03 17:30:30   >> ARG|2|interestRate|double||3.75|
2012-09-03 17:30:36   >> Response from Published Service :
2012-09-03 17:30:36      >> PUB|NOT-OK|-1|Sorry – your team has insufficient permissions to run this service||
2012-09-03 18:05:00 ---
2012-09-03 18:05:00 Calling SOA-Registry with message :
2012-09-03 18:05:00   >> DRC|QUERY-SERVICE|Shcarpo|1183|
2012-09-03 18:05:00   >> SRV|POSTAL|||||
2012-09-03 18:05:03   >> Response from SOA-Registry :
2012-09-03 18:05:03      >> SOA|NOT-OK|-1|Team License Expired||
```

Appendix C - SOA-Registry Error Codes and Messages

Depending on how you call the SOA-Registry and with which SOA message *payload*, you may find that the registry comes back at you with a

```
SOA|NOT-OK|errorCode|errorMessage|
```

response. In order for you to be able to handle and respond to the various errors – you need to know the possibilities of what might be reported. Below is a table of error codes and messages that may come back to you …

Error Code	Error Message	In response to
-1	• Message doesn't contain EOM marker • Invalid segment directive found (<segDirective>) • SOA command <whatever you sent> - UNKNOWN • DRC directive has no embedded SOA command • DRC directive not in first message segment • INF directive not in second message segment • SRV directive not in second message segment	Incorrectly formatted incoming message or unknown SOA command
-2	• DRC/REG-TEAM segment not according to Spec. • DRC/UNREG-TEAM segment not according to Spec. • DRC/QUERY-TEAM segment not according to Spec. • DRC/QUERY-SERVICE segment not according to Spec. • DRC/PUB-SERVICE segment not according to Spec. • EXEC-SERVICE command not processed by SOA-Registry. • INF segment not according to Spec. • INF segment not according to Spec (calling teamName is BLANK). • INF segment not according to Spec (requested tagName is BLANK). • SRV segment not according to Spec (service tagName is BLANK). • SRV segment not according to Spec (service name is BLANK). • SRV segment not according to Spec (security level is BLANK). • SRV segment not according to Spec (numArgs is BLANK). • SRV segment not according to Spec (numResps is BLANK). • SRV segment not according to Spec (calling tagName is BLANK). • SRV segment not according to Spec (All fields after <tagName> must be BLANK). • Tagname (<tagName>) is not valid • ServiceName (<serviceName>) contains invalid characters • Security Level (<securityLevel>) contains invalid value • Number of Arguments (<numArgs>) must be greater than or equal to zero • Number of Responses (<numResp>) must be greater than or equal to one • Service Description contains invalid characters • Service Description too long (needs to be less than 200 characters • ARG Segment #<argNum> - ArgPosition (<whichPos>) is not valid - expected <expectedPos> • ARG Segment #<argNum> - ArgName (<argName>) contains invalid characters • ARG Segment #<argNum> - ArgDatatype (<whichDataType>) is not valid • ARG Segment #<argNum> - ArgMandatoryOptional (<mand-option-value>) is not valid	Incorrectly formed *segment* within message

	• ARG segment (#<argNum>) not according to Spec (argPosition is BLANK). • ARG segment (#<argNum>) not according to Spec (argName is BLANK). • ARG segment (#<argNum>) not according to Spec (argDatatype is BLANK). • ARG segment (#<argNum>) not according to Spec (is the argument mandatory or optional). • RSP Segment #<rspNum> - RespPosition (<whichPos>) is not valid – expected <expectedPos> • RSP Segment #<rspNum> - RespName (<rspName>) contains invalid characters • RSP Segment #<rspNum> - RespDatatype (<whichDataType>) is not valid • RSP segment (#<rspNum>) not according to Spec (rspPosition is BLANK). • RSP segment (#<rspNum>) not according to Spec (rspName is BLANK). • RSP segment (#<rspNum>) not according to Spec (rspDatatype is BLANK). • RSP segment (#<rspNum>) not according to Spec (last response field must be blank). • Inconsistent ARG segments - SRV said <numArgs> ARG segments - found <howMany> such segments. • Inconsistent RSP segments - SRV said <numRsps> RSP segments - found <howMany> such segments. • MCH segment - Service IP Address (<IP-Addr>)is not valid format. • MCH segment - Service IP Address (<IP-Addr>) is not reachable. • MCH Segment - Service Port (<portNum>) not in valid range • MCH segment - Publish Location (<IP-Addr>, port <portNum>) is not accepting connections. • MCH segment not according to Spec (serverIPAddr is BLANK). • MCH segment not according to Spec (serverPort is BLANK).	
-3	• Illegal teamName in INF segment.	Invalid content within the *fields* of the message segment(s)
-4	• No team '<teamName>' (ID : <teamID>) found registered in Dbase • Team '<teamName>' (ID : <teamID>) does not have service <tagName> registered in Dbase • No service <tagName> for team '<teamName>' found in Dbase • Team <teamName> does not have adequate security level to run service <tagName> • Team '<teamName>' (ID : <teamID>) is not registered • Team '<teamName>' (ID : <teamID>) has already published service <tagName> • Team '<teamName>' (ID : <teamID>) is not registered in Dbase • No <tagName> services exist in the Dbase • An error occurred while selecting the <someNumber> <tagName> service in the Dbase • An error occurred while retrieving the number of arguments for <tagName> service (serviceID=<dbaseServiceID>) in the Dbase • An error occurred while retrieving the number of responses for <tagName> service (serviceID=<dbaseServiceID>) in the DBase	SOA message, segments and fields okay – but their content caused some runtime issue
-5	• Error executing SQL=[<sqlStatement>] - error=[<exception thrown>]	Any exception thrown by MSSQLSERVER

It is up to you and your team whether or not (within your Client application) you want to display error messages sent from the SOA-Registry to <u>your</u> user or not. You may wish to take an errorCode / errorMessage and reword it …