

Cancer Trials Support Unit

CTSU – A Service of the National Cancer Institute

OPEN Project - How to use the RandoNode .Net Starter Kit

Revision 02

13 August 2010

Revision Information for the **Cancer Trials Support Unit - OPEN Project - How to use the RandoNode .Net Starter Kit**
Document No. **CTSU/DOC## Rev. 01**

This document was prepared for:

Template Help:

Use the **File→Properties** option to update footer and key values in document.

This document was prepared by:

WESTAT
1650 Research Boulevard
Rockville, Maryland 20850 Phone: (301) 251-1500

Jayan Nair 301-212-2126

Table of Contents

1. Overview	4
2. Audience	4
3. Development Platform and Versions.....	4
4. OPEN portal and RandoNode	4
5. Introduction to RandoNode Starter Kit.....	5
6. How it works.....	5
7. What is included with the installation.....	6
8. CTSU_HOME directory	6
9. Clinical Data files.....	7
10. Metadata files	7
11. How to install the Starter Kit.....	7
12. Opening the solution file and building the projects	10
13. Using the Starter Kit for the first time.....	12
14. Running the RandoNode tester application.....	15
15. Incorporating your business logic.....	15

How to use the RandoNode .Net Starter Kit

1. Overview

This document describes the step by step procedure for installing, building and extending the .Net RandoNode by using the RandoNode Starter Kit.

2. Audience

This document is intended for Cancer Trial Cooperative Groups' IT developers and IT Managers.

3. Development Platform and Versions

This Starter Kit was developed on Microsoft Windows XP operating system and the development platform is Visual Studio 2005 with Microsoft .Net framework version 2.0. Updates will be made to the Starter Kit with future versions of the .Net framework.

3.1. Detailed Requirements

#	Item	Version
1.	Operating System	Microsoft Windows XP or MS Windows Vista
2.	Web Server	Internet Information Service (IIS) version 6 or above
3.	Integrated Development Environment	Visual Studio 2005
4.	Platform	Microsoft .Net framework 2.0 with ASP.Net and C#

Note: Make sure to install (enable) IIS before installing Visual Studio 2005, since ASP.Net engine must be installed after IIS is installed. If this is not the case, then you will need to repair your ASP.Net installation.

4. OPEN portal and RandoNode

OPEN (Oncology Patient Enrollment Network) is an internet portal that will allow authorized users to enroll patients to ongoing cancer treatment clinical trials. The OPEN portal interacts with a set of distributed nodes called RandoNode which are maintained by the lead cooperative groups responsible for originating the cancer trials. RandoNode is a web service which will provide services to the OPEN portal in completing a patient enrollment by generating the patient registration ID as well as randomizing the assignment of the treatment arm. The RandoNode also performs further validation of patient eligibility as well as other bookkeeping and tracking tasks on behalf of the cooperative groups.

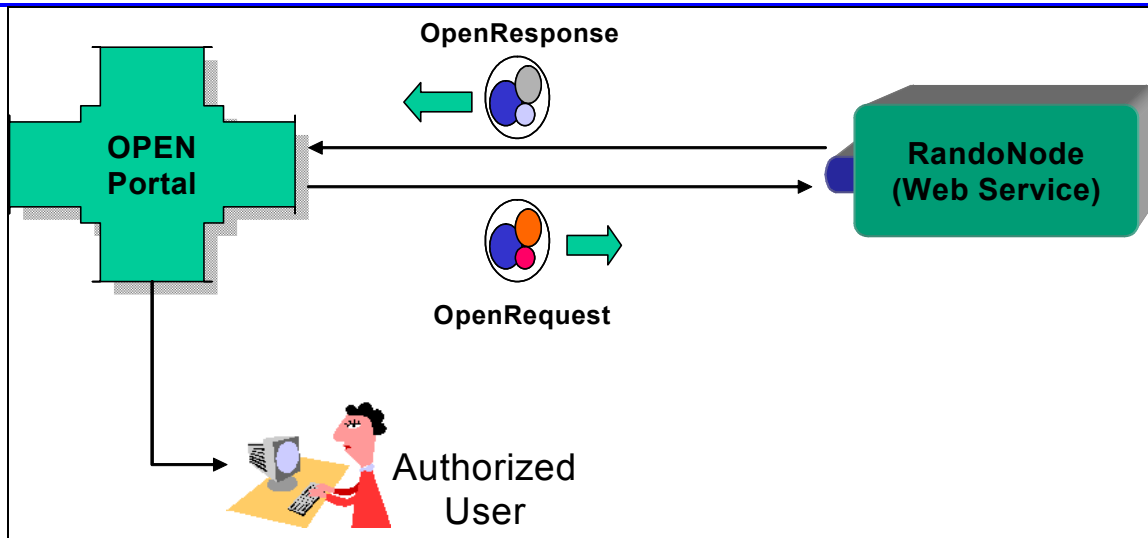


Figure 1: OPEN Portal and RandoNode

5. Introduction to RandoNode Starter Kit

RandoNode Starter Kit is a simple set of applications which includes a demonstration implementation of a RandoNode, framework related DLLs, and a tester application. The RandoNode tester application acts as the proxy for the OPEN portal and can be used to test the RandoNode during the initial development phase.

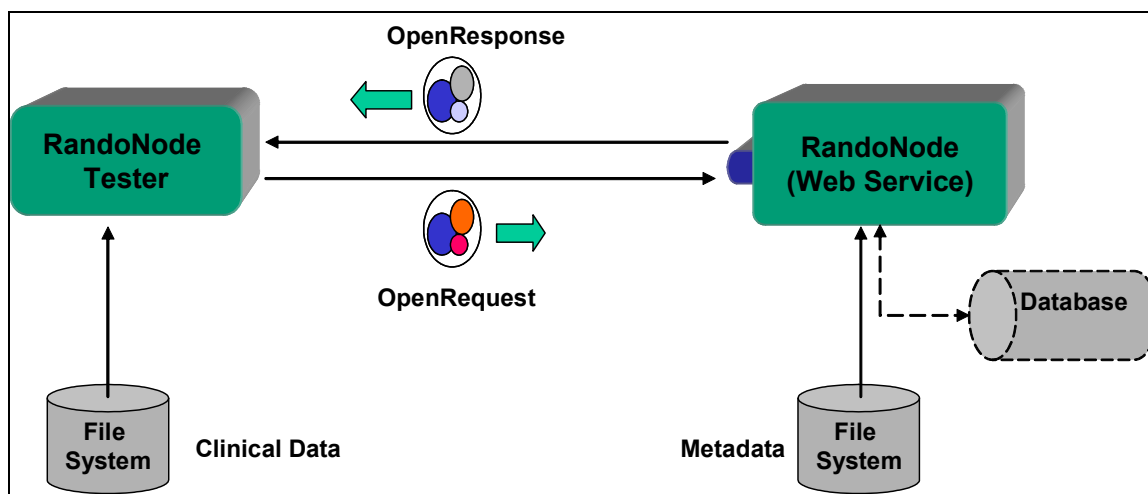


Figure 2: RandoNode Tester acting as a proxy for OPEN portal

6. How it works

As mentioned, the RandoNode tester application acts as a proxy for the OPEN portal and submits patient registrations as though the OPEN portal is submitting it. The registration information to be submitted to the RandoNode is obtained from a Clinical Data XML file stored in the file system where the RandoNode Tester application is running. After submitting the application, the OpenResponse information (the response by the RandoNode to the enrollment request) is displayed by the RandoNode Tester as an XML file on the browser window.

7. Capabilities and features provided through the core libraries of the RandoNode

By using the core libraries (dlls) supplied through the RandoNode starter kit you will get the following capabilities and features.

- ODM data object model.
- Parsing and loading the XML into the domain objects.
- Logging framework for event logging.
- Persistence framework for saving the domain objects to a database.

8. What is included with the installation

RandoNode Starter Kit contains a single installation file named “RandoNodeStarterkit.msi”. When installed, it will create the following projects on the hard drive of the installed machine.

1. RandoNode Project (Web Service) – C# Project
2. RandoNode Tester (OPEN Portal Proxy – Windows Application) – C# Project

In addition, the installation will also copy the following required files to their appropriate directories in the installed machine.

1. CTSU Framework DLLs
 - cdisc.dll – CDISC – ODM auto generated classes
 - node.dll – RandoNode framework classes
 - zdk.dll – CTSU's ZDK software framework
 - persistence.dll – NHibernate persistence related classes along with its dependent dlls as shown below
 - Castle.Core.dll
 - Castle.DynamicProxy2.dll
 - Iesi.Collections.dll
 - log4net.dll
 - NHibernate.dll
 - ODM.dll – The Operational Data Model (ODM) classes which provide access to the study metadata, study data and administrative data associated with a clinical trial.
2. Sample Clinical & Metadata Files
3. API Documentation

9. CTSU_HOME directory

The installation will automatically create a system environment variable called CTSU_HOME which will indicate the default directory to store the clinical and metadata files. The installation will also create directories and subdirectories underneath the CTSU_HOME (C:\CTSU\Applications\) as shown below:

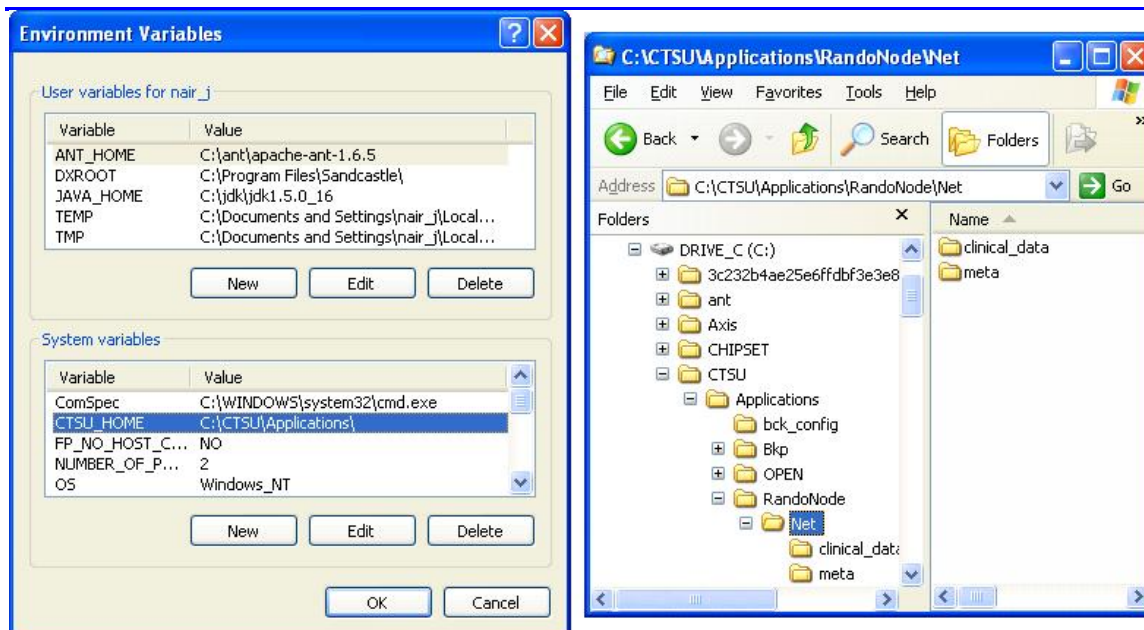


Figure 3: CTSU_HOME Environmental variable and the included folders

The installation will copy over the example clinical data files to the “clinical_data” folder and the metadata files (explained below) to the “meta” folder.

10. Clinical Data files

This file contains example patient enrollment data in the exact format as submitted by the OPEN portal to the RandoNode. The RandoNode Tester application uses this file to load the patient background information required to enroll the patient and to verify the patient’s eligibility.

11. Metadata files

Metadata files contain the enrollment form information which is used to submit the enrollment request by the OPEN portal. There will be a one to one correspondence between an enrollment form and a metadata file. Using the metadata file, the RandoNode developer can extract more details regarding the questions for which the enrollment information (clinical data file) is providing the responses. Various APIs are available through the RandoNode framework DLLs to access the items contained within the clinical and metadata.

12. How to install the Starter Kit

We provide two options for installing the starter kit. For the first time users who are not familiar with setting up the IIS, virtual directories, permissions etc we provide an installation version. After installing using the setup version, you can customize as well as relocate the files once you become more comfortable with the setup.

For users who already began their development using the previous versions of the starter kit, we provide the source code and libraries as a zip archive which can be extracted and selectively applied to your existing project.

12.1. Installing the starter kit from the installation version

Installing the Starter Kit from the installation version is quite straightforward. Just double-click on the file “RandoNodeInstallation.msi” and the installation wizard will guide you through the installation. Until you understand more about creating and configuring Internet Information Service’s (IIS) virtual directories and

creating Web Service projects, it is advised that you accept all the default options provided by the installation program. The following screenshots shows the steps for installing the Starter Kit



Figure 4: Starter Kit Installation Step 1

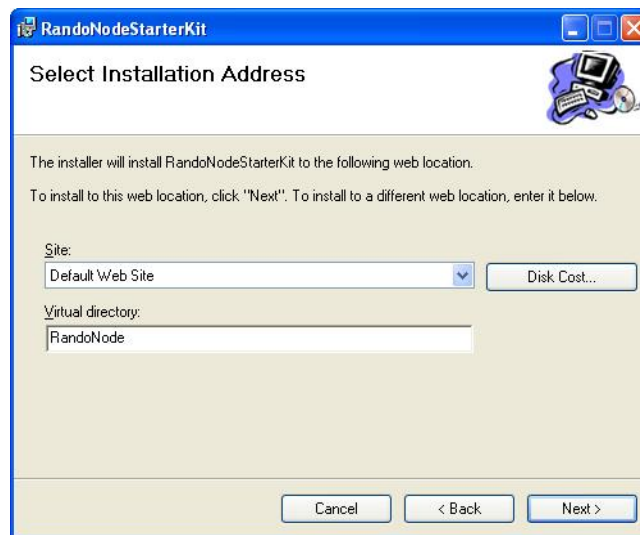


Figure 5: Starter Kit Installation Step 2

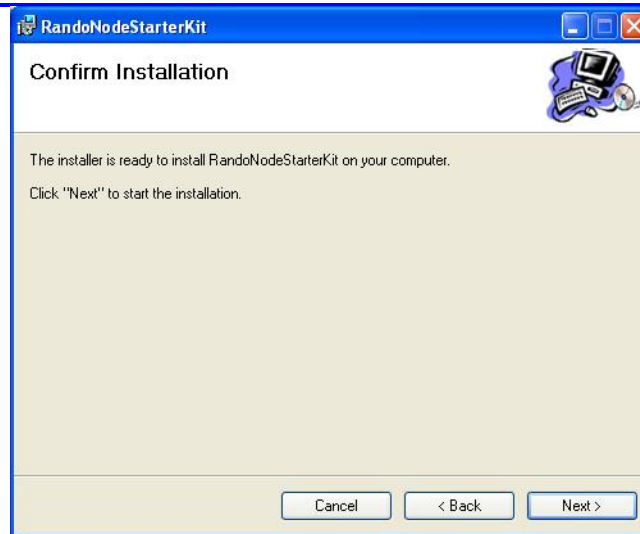


Figure 6: Starter Kit Installation Step 3

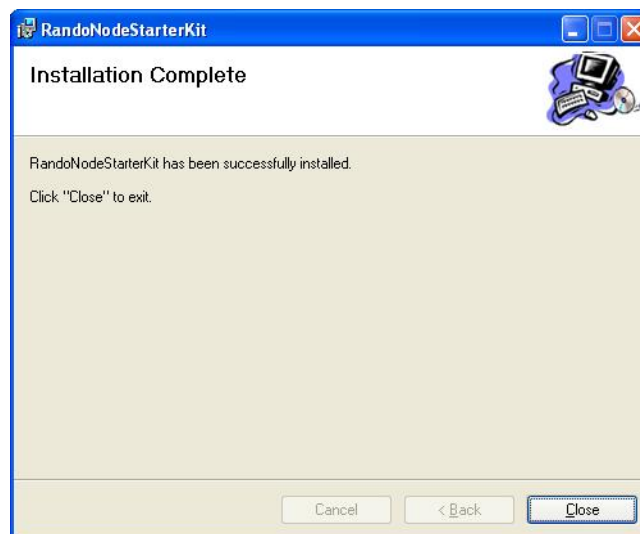


Figure 7: Starter Kit Installation Step 4

12.2. Installation Directory

The RandoNode is a Web Service and hence it is installed as a website in the following folder "C:\inetpub\wwwroot\RandoNode" - which is the default folder for websites using the Internet Information Service (IIS) convention. The RandoNode Tester application is installed in its own subdirectory under the RandoNode main directory as follows: "C:\inetpub\wwwroot\RandoNode\RandoNodeTester". Even though the RandoNodeTester is not a web application it is being installed here just for convenience of being contained in the same location.

Note: Once you are familiar with the web application structure, you can move the installation directory to any location of your choice.

Warning: When installing the Starter Kit from the installation version (from the second time onwards), the original installation will be replaced by the new installation. Hence, if you modify the source files installed by the Starter Kit and if you install the Starter Kit again, then you will lose your modifications. If you like to preserve your modified files, it is strongly recommended that you backup your files before installing the Starter Kit on the top of an existing Starter Kit installation.

12.3. Uninstalling the starter kit

Incase you need to uninstall the starter kit, you can use the installation file "RandoNodeInstallation.msi" itself to uninstall or repair your installation. If you like to uninstall a previous version of the starter kit then there is a utility (bat file) included with the installation as well as the zip archive which is named as "uninstall.bat".

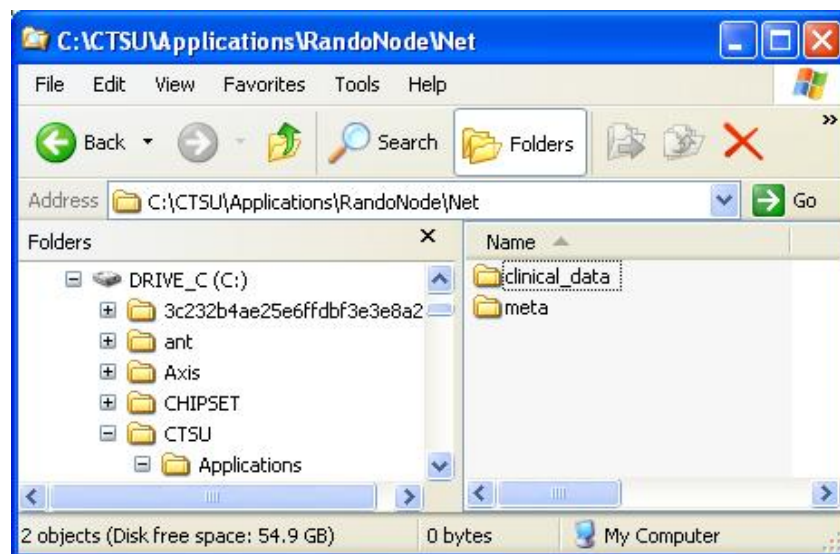
12.4. Installing the starter kit from the zip archive (RandoNodeStarterKit.zip)

This is the recommended method, if you are already using a previous version of the starter kit and would like to selectively replace files without overwriting the existing files.

1. Unzip the files to the directory of your choice.
2. Verify that the RandoNode virtual directory on IIS is pointing to the "RandoNode" source code directory just unzipped from the zip archive.
3. Now copy the metadata folder named "meta" and the clinical data folder named "clinical_data" contained within the "RandoNode" folder to the "CTSUS_HOME" directory you have already defined as the environmental variable. For example if you have defined your CTSU_HOME environmental variable as shown below:

`CTSUS_HOME = C:\CSTU\Applications\`

Then, create the "RandoNode\Net" folders inside "C:\CTSUSApplications\" and then copy the "meta" and "clinical_data" folders to the "Net" folder as shown below:



4. Now, navigate to the "..\RandoNode\RandoNodeTester" folder created while unzipping the archive and open the "RandoNodeTester.sln" file using Visual Studio 2005.
5. Build and run the project.

13. Opening the solution file and building the projects

The installed program is self-contained and is expected to work right out of the box. If you have MS Visual Studio 2005, Integrated Development Environment (IDE) installed on the machine then you can navigate

to the installation directory (or the unzipped zip archive folder) of the RandoNode Tester (C:\inetpub\wwwroot\RandoNode\RandoNodeTester) and double click on the “RandoNodeTester.sln” file to open it within the Visual Studio 2005 IDE.

Once the solution file is opened and loaded, the workspace (Solution Explorer) will look like what is shown below:

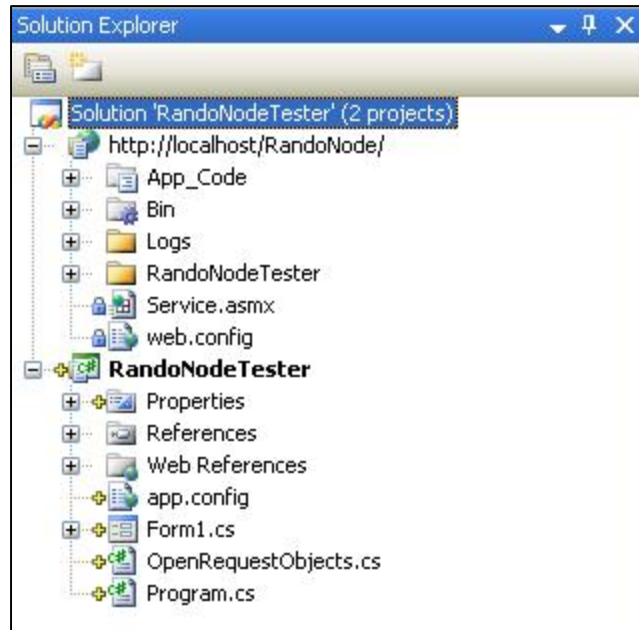


Figure 8: Starter Kit Solution Explorer

You can right click on the solution’s root node, “RandoNodeTester” as shown below and select “Build Solution” on the popup menu to build the project.

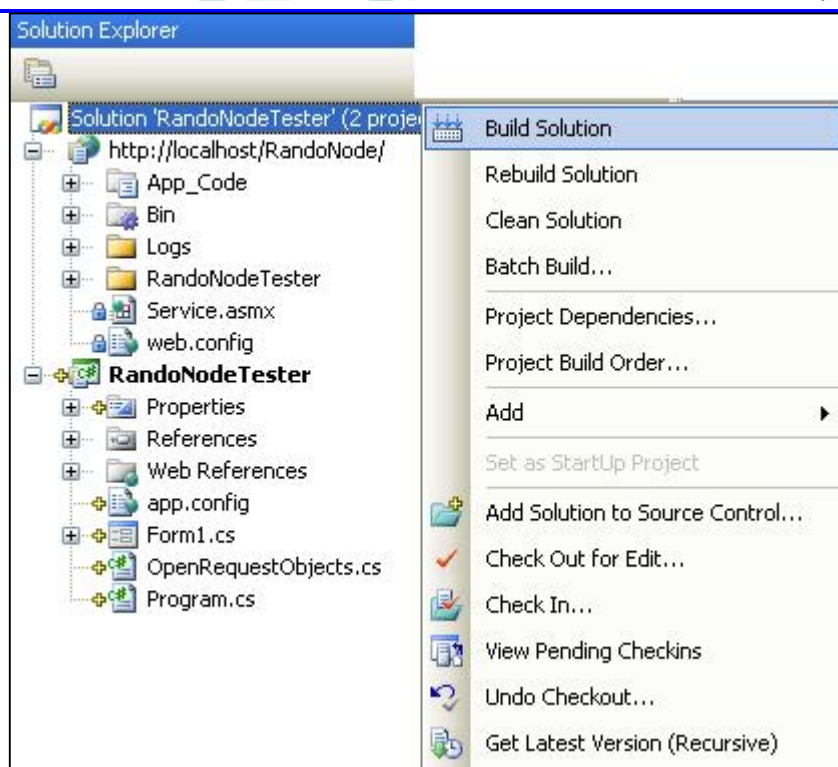


Figure 9: Building the RandoNode Starter kit solution

Once the project is built, use the Run button (Start Debugging or F5) to run the RandoNode Starter Kit.

14. Using the Starter Kit for the first time

After everything is built properly, when you run the RandoNode Tester application, you will see the following dialog.



Figure 10: RandoNode Tester Application

You can click on the link labeled as “Local RandoNode Link” under “Helpful Links” to see if the local RandoNode just installed by the Starter Kit is running or not. When this link is clicked, a new browser window will show up as shown in the following picture.

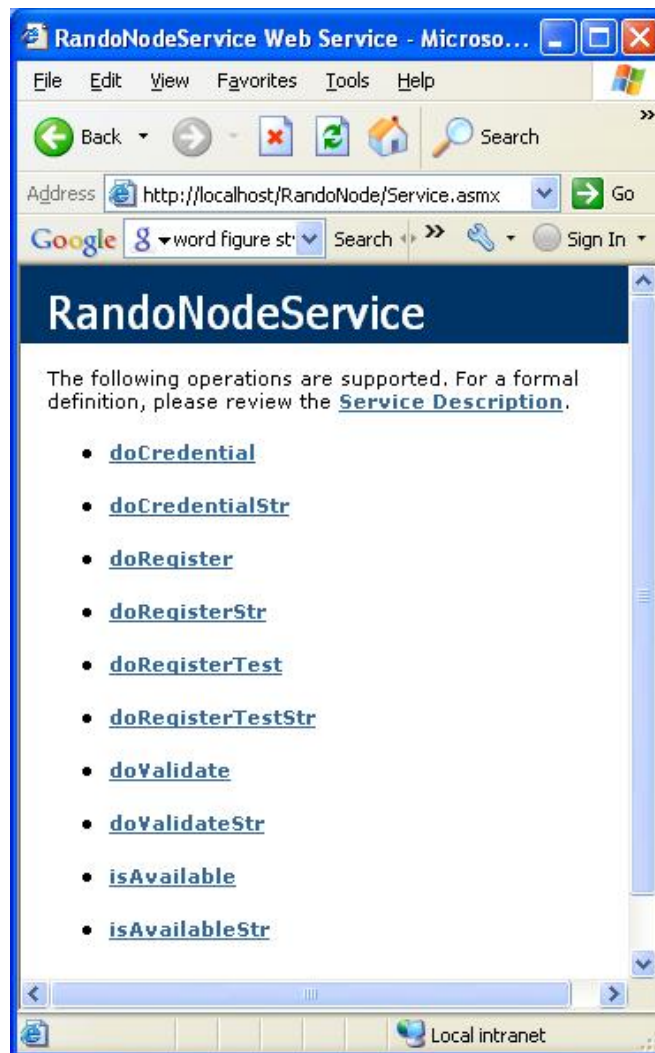


Figure 11: Local RandoNode Service Window

The above browser window displays the local RandoNode service showing all the services (methods) available on the RandoNode. You can click on any service to see its expanded description in SOAP 1.1 Request and Response format as given below for the `doRegister()` method.

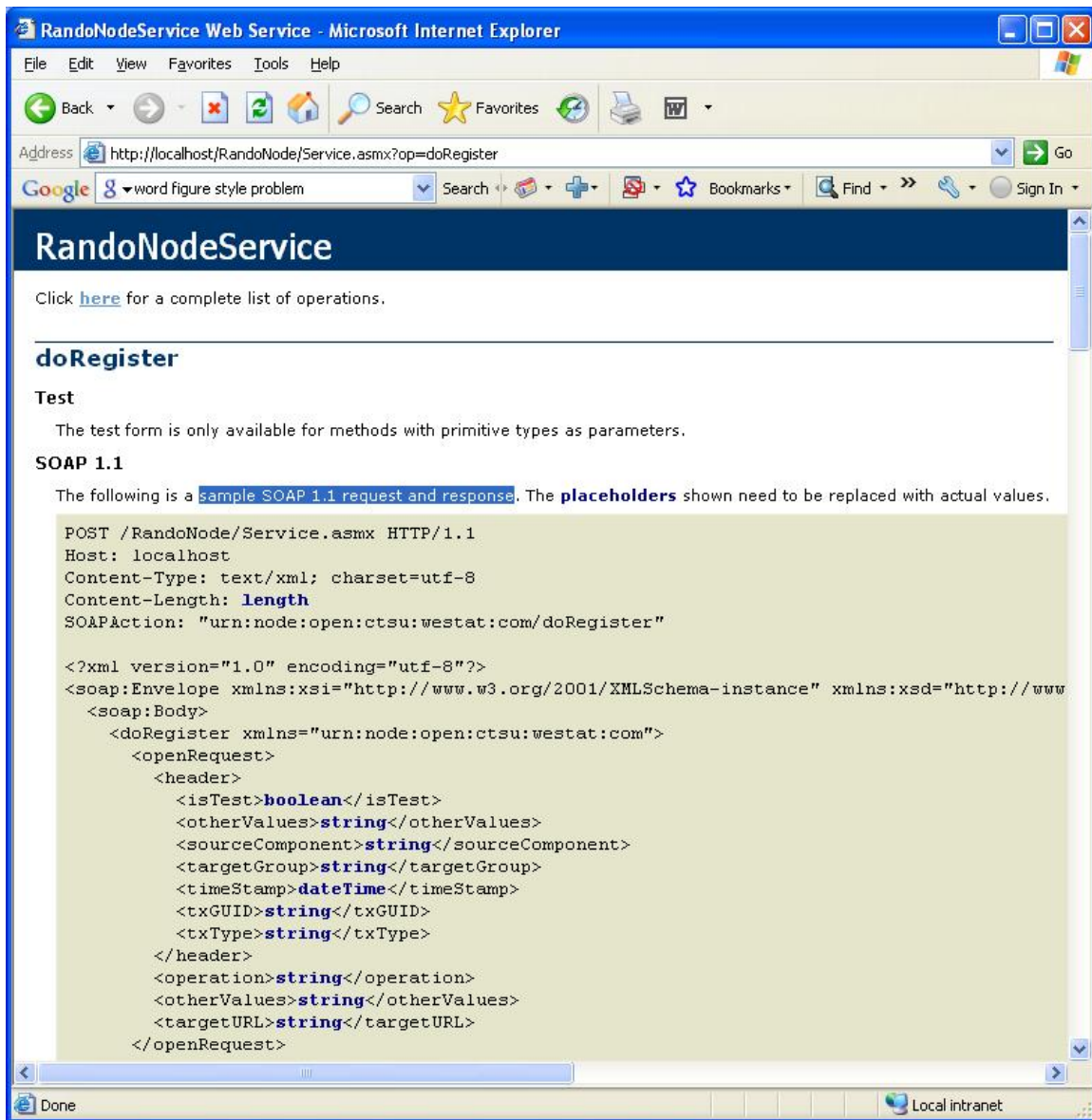


Figure 12: SOAP 1.1 Request and Response for the doRegister() method

If these windows are not showing up, then your installation is not working properly. Please repeat the installation steps. If necessary you may also contact us (CTSUs) for assistance.

15. Running the RandoNode tester application



Figure 13: RandoNode Tester Window

In the above dialog, a sample Clinical Data file name should be pre-filled-in for you. You can display the contents of the clinical data file by using the “View Clinical Data” button. You can also use the browse button (marked by three ellipses) to load a different clinical data file. There will be a corresponding metadata file installed by the installer under the “CTSU_HOME\RandoNode\Net\meta” folder.

Now, you can click on the “Submit” button to load the clinical data file and submit it to the local RandoNode service running on this machine. If everything is working properly the RegistrationResponse data returned by the RandoNode to the RandoNode Tester application will be displayed on the browser window as shown below:

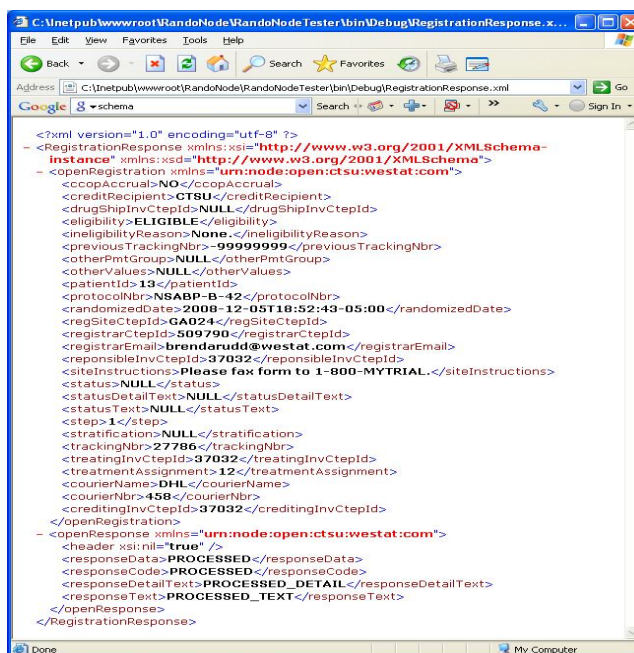


Figure 14: Registration Response message displayed on the browser window as an XML file

16. Configuration settings

You can configure the settings using the “web.config”. The core functionality of the RandoNode supports

logging and persistence. By default persistence and logging will be switched off. You need to update the database connection string on the “web.config” file under the section “hibernate-configuration” and then you can turn on the “persist” key by changing its value to “true” under “appSettings”.

Similarly you can turn on and off the logging feature by changing the value of “WriteServerLog” key to “true”.

16.1. Additional notes on Persistence

We provide a reference implementation for persisting the domain objects to a database, using Oracle as the example database and NHibernate. NHibernate is an Object-relational mapping (ORM) solution for the Microsoft .NET platform. NHibernate uses mapping files to map the attributes of an object to corresponding tables and fields in a database. You will need to create the tables in advance in the database before you can use NHibernate to persist your data.

17. Incorporating your business logic

Once you are able to run the RandoNode Tester application successfully, you can start adding your business logic to your RandoNode.

Essentially you will be modifying the classes included with the RandoNode Web Service project. The RandoNode Tester project will remain generally unmodified unless you want to customize its behavior.

The shells of classes you will be renaming and modifying are included with the Starter Kit installation of the RandoNode Web Service project. The files contained within the example RandoNode Web Service project can be renamed and modified for rapid development and deployment.

To explain the required modifications, the base classes, the derived classes and their inheritance structure are described in the following sections.

17.1. Where to start

The group’s implementation should consist of one factory class which should be derived from the **RandoNodeApp** class, and one registration implementation class which should be derived from the **RegistrationCore** class. Both base classes can be found inside the RandoNode C# project.

The **RandoNodeApp** derived class (e.g. RandoNodeSWOG) is singleton, which should instantiate only once when the Web Service is constructed (see the RandoNodeService constructor for example). This derived class should also override the RandoNodeApp class **getNewRegistration()** method to return a customized derived class of RegistrationCore type. (see RandoNodeSWOG.cs)

The **RegistrationCore** derived class (e.g. RegistrationSWOG) should override the methods that handle the specific Web Service method logic (see RegistrationSWOG.cs).

17.2. How to modify the Web Service example

You can use the following steps to add your business logic. All the files are located within the RandoNode Web Service Project.

1. Rename the RegistrationSWOG class appropriately to reflect your group’s name.
2. Implement business logic methods of the class.
3. Rename the RandoNodeSWOG class appropriately to reflect your group’s name.
4. If you have renamed the RandoNodeSWOG class then, modify the web.config file to specify the new name of the RandoNodeClass as shown below:

```
<appSettings>
  <add key="RandoNodeClass" value="RandoNodeNewName" />
</appSettings>
```

5. Build the web service, and test with RandoNodeTester.

17.3. To implement a new framework from scratch, follow the steps below:

1. Create a new class derived from the RegistrationCore class.

2. Override business logic method of the RegistrationCore.
3. Create a new class derived from the RandoNodeApp class.
4. Override business getNewRegistration() method to return the RegistrationCore descendant.
5. Add/modify the web.config file to specify the value for key **RandoNodeClass**, which should match the name of RandoNodeApp descendant.
6. Build the web service, and test with RandoNodeTester.

17.4. Explanation of the Framework Classes

There are several framework classes that support various services provided by CTSU's RandoNode framework. When you start the development of your RandoNode, the two important classes of interest are (1) RandoNodeApp and (2) RegistrationCore. These classes are explained below.

17.4.1. RandoNodeApp

This class represents the RandoNode application which is the controller class of the RandoNode service. Using this class you can access the RegistrationCore or its derived classes. Additionally this class supports `startup()`, `shutdown()` and `getNode()` methods. For more information, please refer to the RandoNode_API.doc available at <http://www.ctsu.org/open/>.

On your RandoNode, you will have a derived RandoNodeApp class which is shown below, represented by the RandoNodeSWOG class.

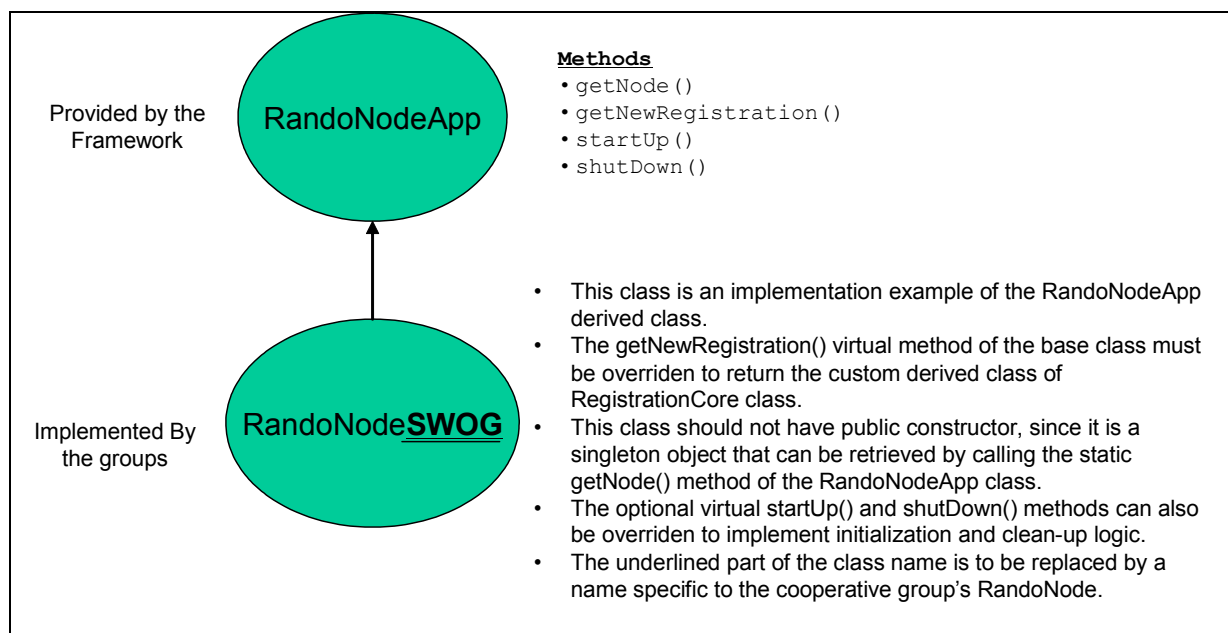


Figure 15: RandoNodeApp Inheritance Diagram

17.4.2. RegistrationCore

As far as patient enrollment is concerned, this is the most important class. This class exposes all the enrollment related methods. You will derive your own class from this class and implement all the registration related functions. A shell implementation of this derived class is included with the RandoNode installed by the Starter Kit, and the class is called RegistrationSWOG as shown below:

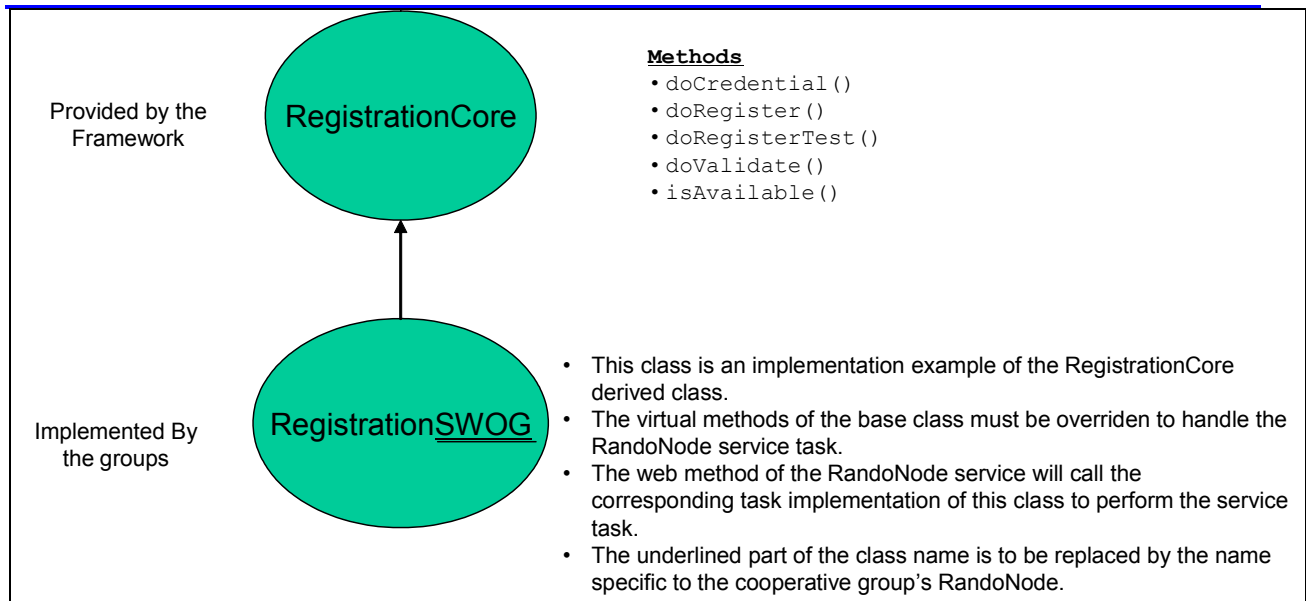


Figure 16: RegistrationCore Inheritance Diagram

17.4.3. Example Implementation

Let us now go to the detail of the `doRegister()` method to see how you can add your own code to customize the behavior of your RandoNode.

In the code listing below, we can see the `doRegister()` method's demonstration implementation. The `RegistrationSWOG` class is shown which the cooperative groups would modify and customize for their purpose.

17.4.3.1. Obtaining the date of birth and age

Inside the `doRegister()` method code listing, the code marked by Section 1, shows how to access the patient's date of birth and age using the item OID value for that particular question.

17.4.3.2. Getting the race of the patient

Inside the same code listing, the code marked by Section 2, shows how to access the patient's race by iterating through the `ItemGroupData` obtained from the form data.

17.4.3.3. Sending back the RegistrationResponse

Inside the `doRegister()` method, the code marked by Section 3, shows how to send back the response after a successful or unsuccessful enrollment.

```
public partial class RegistrationSWOG : RegistrationCore
{
    public RegistrationSWOG (OpenRegistration reg):base(reg) { }

    /// <summary>
    /// This method should be implemented to perform <i>Register</i> task.
    /// </summary>
    /// <param name="tx">Transaction object that contain request
    information.</param>
    /// <param name="response">Response object used to return task
    result.</param>
    /// <returns></returns>
    public override bool doRegister(OpenTxBlock tx, RegistrationResponse
    response)
```

```

{
    #region Sample Implementation
    bool result = base.doRegister(tx, response); //set return to FALSE.
    string dob;
    int age = 0;
    if(doValidate(tx, response))
    {
        //Section 1: Example of getting ItemData by ItemOID
        List<ItemData> list = clinicalDataUtil.getItemData("ID.793");
        if (list.Count > 0)
        {
            dob = list[0].value;

            //Example of calculate age group as a stratum.
            age = ServiceUtil.calAge(DateTime.ParseExact(dob, "yyyyMMdd",
new CultureInfo("en-US")));
        }

        // Section 2: Example of getting race as another stratum, by
searching the ItemData name
        string race = "Not Found";
        bool raceFound = false;
        foreach (KeyValuePair<string, List<ItemGroupData>> kvp in
clinicalDataUtil.getFormData().itemGroupDataMap)
        {
            foreach (ItemGroupData itemGroupData in kvp.Value)
            {
                foreach (ItemData itemData in itemGroupData.itemDataMap)
                {
                    if (itemData.oid == "ID.106" && itemData.value != "")
                    {
                        race = itemData.value;
                        raceFound = true;
                        break;
                    }
                }
            }
            if(raceFound)
            {
                break;
            }
        }
        if (raceFound)
        {
            break;
        }
    }
    Random randNum = new Random();
    if (!raceFound)
    {
        // Do error handling
        response.openRegistration.eligibility = "INELIGIBLE";
        response.openRegistration.ineligibilityReason = "Race is
required for this study";
        response.openRegistration.treatmentAssignment = "None";
        response.openRegistration.patientId = "None";
    }
}

```

```

    }
    else
    {
        // Section 3:
        response.openRegistration.eligibility = "ELIGIBLE";
        response.openRegistration.ineligibilityReason = "None.";
        //perform randomization using race and dob as strata
        string treatArm = randNum.Next(14).ToString();
        response.openRegistration.treatmentAssignment = treatArm;
        response.openRegistration.status = ZAppBlock.SUCCESS;
        response.openRegistration.statusText = "Registration
complete";

        response.openRegistration.statusDetailText = "None";

        // this is to give further information on what
        // is required to complete the process, or further
        // instruction. This can be use to transmit any data.
        response.openRegistration.siteInstructions = "Please send
form to the group address.";
        response.openResponse.responseData = "RESPONSE_DATA";
        response.openResponse.responseCode = ZAppBlock.PROCESSED;
        response.openResponse.responseText = "Request processed";
        response.openResponse.responseDetailText = "Request processed
successfully";

        // Connect to database to assign the next available patient
        id, randomized here as an example.
        string patId = randNum.Next(100).ToString();

        // Set task result into openRegistration and use it as return
        value
        response.openRegistration.patientId = patId;

    }
    return true;
}
return false;
#endregion
}
}

```

18. How Registration Request Objects are Saved into a Designated File

When a registration request is sent over to OPEN, all of its components are objects that cannot physically be seen by the user. This process creates a discrete file in which all of the components of the request are visible.

This task is performed first by translating the request objects into string using `clinicalDataUtil.getRequestXml` method. There is another method `saveRequestXml` inside of `ServicePartial.cs`, which actually writes the input string to an XML file. In order for it to follow through with its task, the value of the key "WriteRequestData" under `appSettings` has to be set to true in the file `web.config`.

A sample code(commented) to call `saveRequestXml` is available within the `doTask` method of `ServicePartial.cs`.

Table 1: SaveRequestXML method details

Method	Parameters		Description
	Input	Output	
saveRequestXml	fileName requestXml	retVal	This method is used to write the input string within the input file name. In order to write the string in a file, the value of "WriteRequestData" in web.config has to be set to true. The code within this method is synchronized to avoid multiple access conflict. The Boolean return value indicates whether the write is successful or not.

```

bool saveRequestXml(string fileName, string requestXml)
{
    bool retVal = false;
    string fullFileName =
System.Web.HttpContext.Current.Server.MapPath("~/Logs/" + fileName);
    if (ConfigurationManager.AppSettings["WriteRequestData"] ==
"true")
    {
        lock (requestXmlLock)
        {
            if (!Directory.Exists(Path.GetDirectoryName(fullFileName)))
            {
                Directory.CreateDirectory(Path.GetDirectoryName(fullFileName));
            }
            StreamWriter writer;
            if (!File.Exists(fullFileName))
            {
                writer = File.CreateText(fullFileName);
            }
            else
            {
                File.Delete(fullFileName);
                writer = File.CreateText(fullFileName);
            }
            writer.WriteLine(requestXml);
            writer.Close();
            retVal = true;
        }
    }
    return retVal;
}

```