**Code Summary**

**HL7ImportWebService.asmx**

The HL7ImportWebService.asmx web service exposes the getHL7Message method to external systems so they can send messages to the Physician Portal application which can then be imported into the Physician Portal database. HL7ImportWebService.asmx is a web service that uses the Simple Object Access Protocol.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Services;

namespace PhysicianPortal2

{

/// <summary>

/// Web service method to receive HL7 formatted String message

/// </summary>

[WebService(Namespace = "http://tempuri.org/")]

[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

[System.ComponentModel.ToolboxItem(false)]

public class HL7ImportWebService : System.Web.Services.WebService

{

[WebMethod]

public void GetHL7Message(String msg)

{

IParser hl7;

hl7 = new HL7Parserv2();

hl7.parseMSG(msg.ToString());

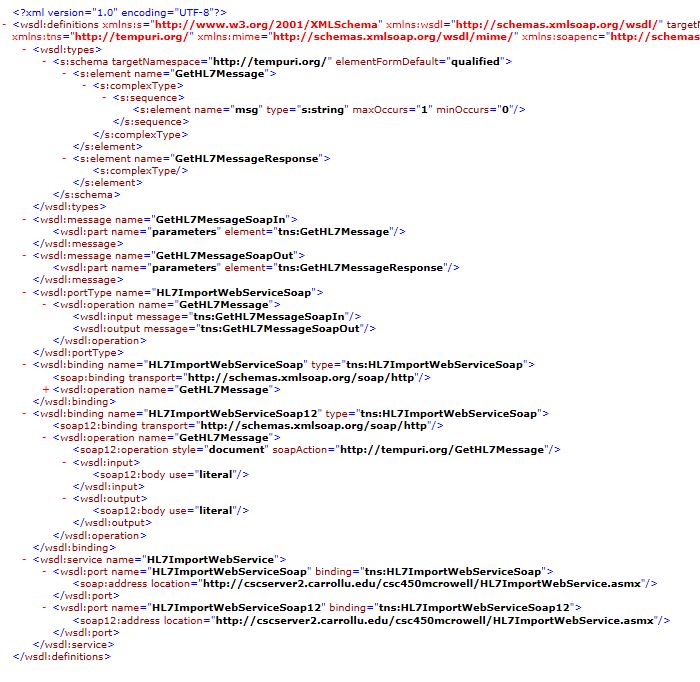
}

}

}

**Web service WSDL XML file**

In order for external systems to be able to successfully send messages to the Physician Portal application, it needs to understand how the Physician Portal application expects to receive messages. This information is shared by providing an XML WSDL file that tells other systems what the Physician Portal application expects to receive.



**IParser.cs and HL7Parserv2.cs**

The IParser.cs interface provides an abstraction to the message import process. This the Physician Portal application extensible and will allow different message types to be added than can be easily imported without the need to modify other parts of the application. The HL7Parserv2.cs class implements the IParser interface and provides the details needed to parse and an understand an HL7 formatted message.

interface IParser

{

void parseMSG(string msg);

}

namespace PhysicianPortal2

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Collections;

using System.IO;

public class HL7Parserv2 : IParser

{

// Public method to receive a string message that is expected to be in HL7 format.

// If one of the HL7 IDs are not identified in the message then it is logged as

// an invalid HL7 message. Otherwise the message is passed to the appropriate method.

public void parseMSG(String msg)

{

String msgID = "";

foreach (string segment in msg.Split(new string[] { "\n", "\r\n" }, StringSplitOptions.RemoveEmptyEntries))

{

//Get field

String[] field = segment.ToString().Split('|');

if (field[0].Equals("MSH"))

{

String[] component = field[8].ToString().Split('^');

msgID = component[0].ToString();

if (msgID.Equals("ADT"))

{

parseADT(msg);

}

else if (msgID.Equals("ORM"))

{

parseOrder(msg);

}

else if (msgID.Equals("ORU"))

{

parseResult(msg);

}

else

{

//invalid HL7 message

INTERFACELOG log = new INTERFACELOG();

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Invalid HL7 message received. \n" + msg;

DataAccess da = new DataAccess();

da.writeToInterfaceLog(log);

}

}

}

}

// This method handles messages with Patient demographic information.

// If the patients exists in the database then the existing record is updated.

// Otherwise a new patient record is added.

private void parseADT(String msg)

{

String ptid = "";

String lastName = "";

String firstName = "";

String dob = "";

String gender = "";

String address = "";

String city = "";

String state = "";

String zip = "";

foreach (string segment in msg.Split(new string[] { "\n", "\r\n" }, StringSplitOptions.RemoveEmptyEntries))

{

String[] field = segment.ToString().Split('|');

if (field[0].Equals("PID"))

{

ptid = field[2].ToString();

String[] nameComponent = field[5].ToString().Split('^');

lastName = nameComponent[0].ToString();

firstName = nameComponent[1].ToString();

dob = field[7].ToString();

gender = field[8].ToString();

String[] addressComponent = field[11].ToString().Split('^');

address = addressComponent[0].ToString();

city = addressComponent[2].ToString();

state = addressComponent[3].ToString();

zip = addressComponent[4].ToString();

}

}

if (!(ptid.Equals("")))

{

PATIENT patient = new PATIENT();

patient.Patient\_ID = ptid;

patient.Last\_Name = lastName;

patient.First\_Name = firstName;

String year = dob.Substring(0, 4);

String month = dob.Substring(4, 2);

String day = dob.Substring(6, 2);

String birthdate = month + "/" + day + "/" + year;

patient.DOB = Convert.ToDateTime(birthdate);

patient.Gender = gender;

patient.Address = address;

patient.City = city;

patient.State = state;

patient.Zip = zip;

DataAccess da = new DataAccess();

if (da.isPatientExist(ptid))

{

da.modifyPatient(patient);

INTERFACELOG log = new INTERFACELOG();

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Patient id " + ptid + " modified.";

da.writeToInterfaceLog(log);

}

else

{

da.addPatient(patient);

INTERFACELOG log = new INTERFACELOG();

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Patient id " + ptid + " added.";

da.writeToInterfaceLog(log);

}

}

else

{

//HL7 error

INTERFACELOG log = new INTERFACELOG();

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Null patient ID received.\n" + msg;

DataAccess da = new DataAccess();

da.writeToInterfaceLog(log);

}

}

// This method handles messages with Patient order information.

// If the order exists in the database then the existing record is updated.

// Otherwise a new order is added.

private void parseOrder(String msg)

{

String orderid = "";

String physicianid = "";

String testcode = "";

String orderdate = "";

String ptid = "";

String siteid = "";

foreach (string segment in msg.Split(new string[] { "\n", "\r\n" }, StringSplitOptions.RemoveEmptyEntries))

{

String[] field = segment.ToString().Split('|');

if (field[0].Equals("PID"))

{

ptid = field[2].ToString();

}

else if (field[0].Equals("PV1"))

{

siteid = field[3].ToString();

}

else if (field[0].Equals("OBR"))

{

orderid = field[2].ToString();

testcode = field[4].ToString();

orderdate = field[7].ToString();

String[] physicianComponent = field[16].ToString().Split('^');

physicianid = physicianComponent[0].ToString();

}

}

if (!(orderid.Equals("")))

{

ORDER order = new ORDER();

order.Order\_ID = orderid;

order.Physician\_ID = physicianid;

order.Test\_Code = testcode;

String year = orderdate.Substring(0, 4);

String month = orderdate.Substring(4, 2);

String day = orderdate.Substring(6, 2);

String newdate = month + "/" + day + "/" + year;

order.Order\_Date = Convert.ToDateTime(newdate);

order.Order\_Status = "Pending";

order.Patient\_ID = ptid;

order.Site\_ID = siteid;

DataAccess da = new DataAccess();

if (da.isOrderExist(orderid))

{

da.modifyOrder(order);

INTERFACELOG log = new INTERFACELOG();

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Order id " + orderid + " modified.";

da.writeToInterfaceLog(log);

}

else

{

da.addOrder(order);

INTERFACELOG log = new INTERFACELOG();

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Order id " + orderid + " added.";

da.writeToInterfaceLog(log);

}

}

else

{

//HL7 error

INTERFACELOG log = new INTERFACELOG();

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Null order ID received.\n" + msg;

DataAccess da = new DataAccess();

da.writeToInterfaceLog(log);

}

}

// This method handles messages with result information.

// If the result exists in the database then the existing record is updated.

// Otherwise a new result is added.

// The method will also update the order status of the order record.

private void parseResult(String msg)

{

String resultid = "";

String resultdate = "";

String resultedby = "";

String resultreport = "";

String orderid = "";

foreach (string segment in msg.Split(new string[] { "\n", "\r\n" }, StringSplitOptions.RemoveEmptyEntries))

{

String[] field = segment.ToString().Split('|');

if (field[0].Equals("OBR"))

{

resultid = field[2].ToString();

orderid = field[3].ToString();

resultdate = field[7].ToString();

resultedby = field[11].ToString();

}

else if (field[0].Equals("OBX"))

{

resultreport += field[5].ToString() + ", ";

}

}

if (!(resultid.Equals("")))

{

DataAccess da = new DataAccess();

ORDER order = da.getORDERByOrderID(orderid);

if (!(order == null))

{

RESULT result = new RESULT();

result.Result\_ID = resultid;

String year = resultdate.Substring(0, 4);

String month = resultdate.Substring(4, 2);

String day = resultdate.Substring(6, 2);

String newdate = month + "/" + day + "/" + year;

result.Result\_Date = Convert.ToDateTime(newdate);

result.Resulted\_By = resultedby;

result.Result\_Report = resultreport;

result.Order\_ID = orderid;

INTERFACELOG log = new INTERFACELOG();

if (da.isResultExist(resultid))

{

da.modifyResult(result);

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Result id " + resultid + " modified.";

da.writeToInterfaceLog(log);

}

else

{

da.addResult(result);

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Result id " + resultid + " added.";

da.writeToInterfaceLog(log);

}

order.Order\_Status = "Resulted";

da.modifyOrder(order);

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Order id " + orderid + " modified.";

da.writeToInterfaceLog(log);

}

else

{

INTERFACELOG log = new INTERFACELOG();

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Result contained an order ID that does not exist.\n Results can only be applied to an existing order. \n" + msg;

da.writeToInterfaceLog(log);

}

}

else

{

//HL7 error

INTERFACELOG log = new INTERFACELOG();

log.Date\_Time = DateTime.Now;

log.Log\_Entry = "Null result ID received.\n" + msg;

DataAccess da = new DataAccess();

da.writeToInterfaceLog(log);

}

}

}

}

**AccountMembershipProvider.cs and AccountRoleProvider.cs**

The AccountMembershipProvider and AccountRoleProvider classes, which .NET uses to manage security and users/roles, were overridden to provide custom functionality to allow the Physician Portal application to define how the users and roles should be managed. The custom AccountMembershipProvider and AccountRoleProvider classes will rely and the Physician Portal database to authenticate users and roles.

public override string GetPassword(string username, string answer)

{

DataAccess da = new DataAccess();

return da.getUserPassword(username);

return "";

}

public override System.Web.Security.MembershipUser GetUser(string username, bool userIsOnline)

{

DataAccess da = new DataAccess();

if (da.isValidUser(username))

{

User newUser = new User(username);

return newUser;

}

else

{

MembershipUser newMember = new MembershipUser(

providerName: "AccountMembershipProvider",

name: "",

providerUserKey: null,

email: "",

passwordQuestion: "",

comment: "",

isApproved: false,

isLockedOut: false,

creationDate: DateTime.UtcNow,

lastLoginDate: DateTime.UtcNow,

lastActivityDate: DateTime.UtcNow,

lastPasswordChangedDate: DateTime.UtcNow,

lastLockoutDate: DateTime.UtcNow);

return newMember;

}

}

public override bool ValidateUser(string username, string password)

{

DataAccess da = new DataAccess();

if (da.isValidUser(username) && da.getUserPassword(username).Equals(password))

{

return true;

}

else

{

return false;

}

}

// Determine roles based on values from the SQL database.

public override string[] GetRolesForUser(string username)

{

String[] roles;

DataAccess da = new DataAccess();

if (da.isUserAdmin(username))

{

roles = new String[] { "Admin" };

}

else if (da.isUserPhysician(username))

{

roles = new String[] { "Physician" };

}

else

{

roles = new String[] { "User" };

}

return roles;

}

public override bool IsUserInRole(string username, string roleName)

{

DataAccess da = new DataAccess();

if (da.isUserAdmin(username) && roleName.Equals("Admin"))

{

return true;

}

else if (da.isUserPhysician(username) && roleName.Equals("Physician"))

{

return true;

}

else if (!(da.isUserPhysician(username)) && !(da.isUserAdmin(username)) && roleName.Equals("User"))

{

return true;

}

else

{

return false;

}

}

**Physician.aspx.cs**

The Physician.aspx.cs code behind class uses the identity of the logged in user to display a custom list of only those patients that are being seen by the physician that is logged into the Physician Portal application. The results are displayed in a three layered nested gridview control that allows the user to drill down from a list of patients to orders for each patient and finally to the result report for each order.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

namespace PhysicianPortal2.PhysicianPages

{

public partial class Physician : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

if (!IsPostBack)

{

DataAccess da = new DataAccess();

String physID = da.getPhysicianIDByUserID(Page.User.Identity.Name);

gvPatient.DataSource = da.getPatientsByPhysicianID(physID);

gvPatient.DataBind();

}

}

protected void gvPatient\_OnRowDataBound(object sender, GridViewRowEventArgs e)

{

if (e.Row.RowType == DataControlRowType.DataRow)

{

Label lblPtID = (Label)e.Row.FindControl("lblPtID");

GridView gvOrders = (GridView)e.Row.FindControl("gvOrders");

string txtptid = lblPtID.Text;

DataAccess da = new DataAccess();

gvOrders.DataSource = da.getPatientOrdersByPatientID(txtptid);

gvOrders.DataBind();

}

}

protected void gvOrders\_OnRowDataBound(object sender, GridViewRowEventArgs e)

{

if (e.Row.RowType == DataControlRowType.DataRow)

{

Label lblOrderID = (Label)e.Row.FindControl("lblOrderID");

GridView gvResult = (GridView)e.Row.FindControl("gvResult");

string txtorderid = lblOrderID.Text;

DataAccess da = new DataAccess();

gvResult.DataSource = da.getResultListByOrderID(txtorderid);

gvResult.DataBind();

}

}

}

}

**Patient.cs**

A Patient.cs class was created to represent the PATIENT table in the database and to implement the IComparable interface to allow patients to be sorted by first and last name. The User.cs class represents the USER table in the database. It implements the IComparable interface and extends MemberShipUser. Extending MembershipUser allows the data from the user table to be used in the .NET management of user authentication and roles.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace PhysicianPortal2

{

public class Patient : IComparable<Patient>

{

public string ptid { get; set; }

public string lastName { get; set; }

public string firstName { get; set; }

public string dob { get; set; }

public string address { get; set; }

public string city { get; set; }

public string state { get; set; }

public string zip { get; set; }

#region IComparable<DataFile> Members

public int CompareTo(Patient other)

{

return String.Compare(this.ptid, other.ptid);

}

#endregion

public override string ToString()

{

return "ID: " + ptid + "\n" +

"Patient Name: " + lastName + ", " + firstName + "\n" +

"DOB: " + dob + "\n" +

"Address: " + address + "\n"

+ city + ", " + state + " " + zip + "\n";

}

public override int GetHashCode()

{

return this.ptid.GetHashCode();

}

}

}

namespace PhysicianPortal2

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

public class User : System.Web.Security.MembershipUser, IComparable<User>

{

public string username { get; set; }

public string password { get; set; }

public bool isPhysician { get; set; }

public string userLogin { get; set; }

public User()

{

}

public User(String name)

: base(

providerName: "AccountMembershipProvider",

name: name,

providerUserKey: null,

email: "",

passwordQuestion: "",

comment: "",

isApproved: true,

isLockedOut: false,

creationDate: DateTime.UtcNow,

lastLoginDate: DateTime.UtcNow,

lastActivityDate: DateTime.UtcNow,

lastPasswordChangedDate: DateTime.UtcNow,

lastLockoutDate: DateTime.UtcNow)

{

userLogin = name;

}

#region IComparable<User> Members

public int CompareTo(User other)

{

return String.Compare(this.username, other.username);

}

#endregion

}

}

**DataAccess.cs**

The DataAccess.cs class provides all of the Linq queries to represent the business logic that deliver the necessary datasets. The view classes will call the methods in the DataAccess class so they are not ties directly to the data layer. This provides a separation between the data and view layers so changes can be made in one layer without having an impact on the other.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace PhysicianPortal2

{

public class DataAccess

{

private LinqToAllDataDataContext linqDat = new LinqToAllDataDataContext();

public List<USER> getListofAllUsers()

{

List<USER> uList = new List<USER>();

var u = from usr in linqDat.USERs select usr;

foreach (var v in u)

{

USER newU = new USER();

newU.User\_Name = v.User\_Name;

newU.Password = v.Password;

newU.Is\_Physician = v.Is\_Physician;

uList.Add(newU);

}

return uList;

}

public bool isValidUser(String name)

{

var users = (from dat in linqDat.USERs

where dat.User\_Name.Equals(name)

select new { dat }).ToList();

if (users.Count > 0)

{

if (users.First().dat.Is\_Active)

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

public String getUserPassword(String name)

{

var users = (from dat in linqDat.USERs

where dat.User\_Name.Equals(name)

select new { dat }).ToList();

if (users.Count > 0)

{

return users.First().dat.Password;

}

else

{

return null;

}

}

public bool isUserAdmin(String name)

{

var users = (from dat in linqDat.USERs

where dat.User\_Name.Equals(name)

select new { dat }).ToList();

if (users.Count > 0)

{

return users.First().dat.Is\_Admin;

}

else

{

return false;

}

}

public bool isUserPhysician(String name)

{

var users = (from dat in linqDat.USERs

where dat.User\_Name.Equals(name)

select new { dat }).ToList();

if (users.Count > 0)

{

return users.First().dat.Is\_Physician;

}

else

{

return false;

}

}

public void addPatient(Patient patient)

{

PATIENT newPatient = new PATIENT();

newPatient.Patient\_ID = patient.ptid;

newPatient.Last\_Name = patient.lastName;

newPatient.First\_Name = patient.firstName;

DateTime dt = Convert.ToDateTime(patient.dob);

newPatient.DOB = dt;

newPatient.Address = patient.address;

newPatient.City = patient.city;

newPatient.State = patient.state;

newPatient.Zip = patient.zip;

linqDat.PATIENTs.InsertOnSubmit(newPatient);

linqDat.SubmitChanges();

}

public Patient getPatientByID(String id)

{

var patients = (from dat in linqDat.PATIENTs

where dat.Patient\_ID.Equals(id)

select new { dat }).ToList();

if (patients.Count > 0)

{

Patient patient = new Patient();

patient.ptid = patients.First().dat.Patient\_ID;

patient.firstName = patients.First().dat.First\_Name;

patient.lastName = patients.First().dat.Last\_Name;

patient.address = patients.First().dat.Address;

patient.city = patients.First().dat.City;

patient.state = patients.First().dat.State;

patient.zip = patients.First().dat.Zip;

patient.dob = patients.First().dat.DOB.ToString();

return patient;

}

else

{

return null;

}

}

public List<ORDER> getPatientOrdersByPatientID(String id)

{

List<ORDER> orderList = new List<ORDER>();

var ord = from dat in linqDat.PATIENTs

join orders in linqDat.ORDERs on dat.Patient\_ID equals orders.Patient\_ID

where dat.Patient\_ID.Equals(id)

select new { orders };

foreach (var v in ord)

{

ORDER o = new ORDER();

o.Order\_ID = v.orders.Order\_ID;

o.Physician\_ID = v.orders.Physician\_ID;

o.Test\_Code = v.orders.Test\_Code;

o.Order\_Date = v.orders.Order\_Date;

o.Order\_Status = v.orders.Order\_Status;

orderList.Add(o);

}

return orderList;

}

public List<PATIENT> getAllPatients()

{

List<PATIENT> patientList = new List<PATIENT>();

var p = from pat in linqDat.PATIENTs select pat;

foreach (var v in p)

{

PATIENT newPat = new PATIENT();

newPat.Patient\_ID = v.Patient\_ID;

newPat.Last\_Name = v.Last\_Name;

newPat.First\_Name = v.First\_Name;

newPat.DOB = v.DOB;

newPat.Address = v.Address;

newPat.City = v.City;

newPat.State = v.State;

newPat.Zip = v.Zip;

patientList.Add(newPat);

}

return patientList;

}

public List<PATIENT> getPatientsByPhysicianID(String id)

{

List<PATIENT> patientList = new List<PATIENT>();

var dat = from pat in linqDat.PATIENTs

join orders in linqDat.ORDERs on pat.Patient\_ID equals orders.Patient\_ID

where orders.Physician\_ID.Equals(id)

select new { pat};

foreach (var v in dat)

{

PATIENT newPat = new PATIENT();

newPat.Patient\_ID = v.pat.Patient\_ID;

newPat.Last\_Name = v.pat.Last\_Name;

newPat.First\_Name = v.pat.First\_Name;

newPat.DOB = v.pat.DOB;

newPat.Address = v.pat.Address;

newPat.City = v.pat.City;

newPat.State = v.pat.State;

newPat.Zip = v.pat.Zip;

patientList.Add(newPat);

}

return patientList;

}

public RESULT getResultByOrderID(String id)

{

var result = (from res in linqDat.RESULTs

where res.Order\_ID.Equals(id)

select new { res }).ToList();

if (result.Count > 0)

{

return result.First().res;

}

else

{

return null;

}

}

public List<RESULT> getResultListByOrderID(String id)

{

var result = (from res in linqDat.RESULTs

where res.Order\_ID.Equals(id)

select new { res }).ToList();

List<RESULT> resultList = new List<RESULT>();

foreach (var v in result)

{

RESULT newRes = new RESULT();

newRes.Result\_ID = v.res.Result\_ID;

newRes.Order\_ID = v.res.Order\_ID;

newRes.Result\_Date = v.res.Result\_Date;

newRes.Resulted\_By = v.res.Resulted\_By;

newRes.Result\_Report = v.res.Result\_Report;

resultList.Add(newRes);

}

return resultList;

}

public String getPhysicianIDByUserID(String id)

{

var result = (from res in linqDat.PHYSICIANs

where res.User\_Name.Equals(id)

select new { res }).ToList();

if (result.Count > 0)

{

return result.First().res.Physician\_ID;

}

else

{

return null;

}

}

public bool isPatientExist(String id)

{

var patients = (from dat in linqDat.PATIENTs

where dat.Patient\_ID.Equals(id)

select new { dat }).ToList();

if (patients.Count > 0)

{

return true;

}

else

{

return false;

}

}

public void addPatient(PATIENT patient)

{

linqDat.PATIENTs.InsertOnSubmit(patient);

linqDat.SubmitChanges();

}

public void modifyPatient(PATIENT patient)

{

PATIENT modPatient = getPATIENTByPtID(patient.Patient\_ID.ToString());

modPatient.Patient\_ID = patient.Patient\_ID;

modPatient.Last\_Name = patient.Last\_Name;

modPatient.First\_Name = patient.First\_Name;

modPatient.DOB = patient.DOB;

modPatient.Address = patient.Address;

modPatient.City = patient.City;

modPatient.State = patient.State;

modPatient.Zip = patient.Zip;

modPatient.Gender = patient.Gender;

linqDat.SubmitChanges();

}

public PATIENT getPATIENTByPtID(String id)

{

var patients = (from dat in linqDat.PATIENTs

where dat.Patient\_ID.Equals(id)

select new { dat }).ToList();

if (patients.Count > 0)

{

return patients.First().dat;

}

else

{

return null;

}

}

public bool isOrderExist(String id)

{

var orders = (from dat in linqDat.ORDERs

where dat.Order\_ID.Equals(id)

select new { dat }).ToList();

if (orders.Count > 0)

{

return true;

}

else

{

return false;

}

}

public void addOrder(ORDER order)

{

linqDat.ORDERs.InsertOnSubmit(order);

linqDat.SubmitChanges();

}

public void modifyOrder(ORDER order)

{

ORDER modOrder = getORDERByOrderID(order.Order\_ID.ToString());

modOrder.Order\_ID = order.Order\_ID;

modOrder.Physician\_ID = order.Physician\_ID;

modOrder.Test\_Code = order.Test\_Code;

modOrder.Order\_Date = order.Order\_Date;

modOrder.Order\_Status = order.Order\_Status;

modOrder.Patient\_ID = order.Patient\_ID;

modOrder.Site\_ID = order.Site\_ID;

linqDat.SubmitChanges();

}

public ORDER getORDERByOrderID(String id)

{

var orders = (from dat in linqDat.ORDERs

where dat.Order\_ID.Equals(id)

select new { dat }).ToList();

if (orders.Count > 0)

{

return orders.First().dat;

}

else

{

return null;

}

}

public bool isResultExist(String id)

{

var results = (from dat in linqDat.RESULTs

where dat.Result\_ID.Equals(id)

select new { dat }).ToList();

if (results.Count > 0)

{

return true;

}

else

{

return false;

}

}

public void addResult(RESULT result)

{

linqDat.RESULTs.InsertOnSubmit(result);

linqDat.SubmitChanges();

}

public void modifyResult(RESULT result)

{

RESULT modResult = getRESULTByResultID(result.Result\_ID.ToString());

modResult.Result\_ID = result.Result\_ID;

modResult.Result\_Date = result.Result\_Date;

modResult.Resulted\_By = result.Resulted\_By;

modResult.Result\_Report = result.Result\_Report;

modResult.Order\_ID = result.Order\_ID;

linqDat.SubmitChanges();

}

public RESULT getRESULTByResultID(String id)

{

var results = (from dat in linqDat.RESULTs

where dat.Result\_ID.Equals(id)

select new { dat }).ToList();

if (results.Count > 0)

{

return results.First().dat;

}

else

{

return null;

}

}

public void writeToInterfaceLog(INTERFACELOG log)

{

linqDat.INTERFACELOGs.InsertOnSubmit(log);

linqDat.SubmitChanges();

}

public List<INTERFACELOG> getInterfaceLog()

{

List<INTERFACELOG> logList = new List<INTERFACELOG>();

var l = from log in linqDat.INTERFACELOGs select log;

foreach (var v in l)

{

INTERFACELOG newLog = new INTERFACELOG();

newLog.Date\_Time = v.Date\_Time;

newLog.Log\_Entry = v.Log\_Entry;

logList.Add(newLog);

}

return logList;

}

public void addSite(SITE site)

{

linqDat.SITEs.InsertOnSubmit(site);

linqDat.SubmitChanges();

}

public void modifySite(SITE site)

{

SITE modSite = getSiteByID(site.Site\_ID.ToString());

modSite.Site\_ID = site.Site\_ID;

modSite.Name = site.Name;

modSite.Address = site.Address;

modSite.City = site.City;

modSite.State = site.State;

modSite.Zip = site.Zip;

linqDat.SubmitChanges();

}

public SITE getSiteByID(String id)

{

var sites = (from dat in linqDat.SITEs

where dat.Site\_ID.Equals(id)

select new { dat }).ToList();

if (sites.Count > 0)

{

return sites.First().dat;

}

else

{

return null;

}

}

public void addTestCode(TEST test)

{

linqDat.TESTs.InsertOnSubmit(test);

linqDat.SubmitChanges();

}

public void modifyTestCode(TEST test)

{

TEST modTest = getTestCodeByID(test.Test\_Code.ToString());

modTest.Test\_Code = test.Test\_Code;

modTest.Description = test.Description;

linqDat.SubmitChanges();

}

public TEST getTestCodeByID(String id)

{

var tests = (from dat in linqDat.TESTs

where dat.Test\_Code.Equals(id)

select new { dat }).ToList();

if (tests.Count > 0)

{

return tests.First().dat;

}

else

{

return null;

}

}

public List<SITE> getListofAllSites()

{

List<SITE> sList = new List<SITE>();

var s = from site in linqDat.SITEs select site;

foreach (var v in s)

{

SITE newS = new SITE();

newS.Site\_ID = v.Site\_ID;

newS.Name = v.Name;

newS.Address = v.Address;

newS.City = v.City;

newS.State = v.State;

newS.Zip = v.Zip;

sList.Add(newS);

}

return sList;

}

public List<TEST> getListofAllTests()

{

List<TEST> tList = new List<TEST>();

var t = from test in linqDat.TESTs select test;

foreach (var v in t)

{

TEST newT = new TEST();

newT.Test\_Code = v.Test\_Code;

newT.Description = v.Description;

tList.Add(newT);

}

return tList;

}

public List<TestCodeReport> getTestCodeUseage()

{

var testcodes = from order in linqDat.ORDERs

join test in linqDat.TESTs on order.Test\_Code equals test.Test\_Code

group new

{

order,

test

} by new

{

test.Test\_Code

} into g

select new

{

testCount = g.Count(),

testCode = g.Key.Test\_Code

};

if (testcodes.Count() > 0)

{

List<TestCodeReport> tList = new List<TestCodeReport>();

foreach (var v in testcodes)

{

TestCodeReport tr = new TestCodeReport();

tr.tetCode = v.testCode;

tr.useageCount = v.testCount;

tList.Add(tr);

}

return tList;

}

else

{

return null;

}

}

}

}