Paper TS07

Using Web Services to exchange information via XML

Edward Foster, Oxford Pharmaceutical Sciences, UK

ABSTRACT

Web Services have evolved over the past 4 years and are a key component of Service Oriented Architecture (SOA). Web Services are units of code that can be called via HTTP requests in the same way that traditional websites can. The fundamental difference between them is that web services do not normally send HTML in response to a request but instead emit XML. The use of HTTP and XML gives web services advantages of platform and technology independence and allow web services to be set up to work across any intranet or even the internet. .NET is a powerful framework created by Microsoft that contains a host of classes that can be used to create powerful web and Windows applications as well as web services. This paper will outline the core elements of web services, their use and also how .NET can be used to create a web service.

INTRODUCTION

While there are similarities between traditional Web Applications and Web Services the main function of web services is to allow a client to perform remote method calls over the HTTP protocol rather than serve HTML content to a web browser for display. Historically, accessing remote objects required the implementation of platform and language specific protocols such as DCOM or Java RMI. The main problem with these approaches is that while these technologies work well in a homogeneous environment between like for like systems attempts to create truly distributed solutions could run into problems. The main reason for this is that these technologies often worked using proprietary wire formats that did not work well on different operating systems or in conjunction with other technologies. This is where web services have the advantage. Web services use the HTTP transport protocol which is perhaps the most ubiquitous in the world as it is the basis of the internet! Using HTTP means that web services can be hosted within web servers and as such can be available across a company in the same way an intranet sit. But why stop at the company boundary? HTTP is the basis of the internet so why not be available to the world! This in fact is a key application of web services as it allows Business to Business (B2B) communication.

The second element of web services is XML. XML in essence is just a well formed text document that can describe data structures using simple tags. Being a text document XML is therefore totally platform and technology independent. Even if you are trying to integrate a .NET application with a Java program, though XML data exchange it becomes easy because both technologies can read text and can understand what is being passed to them from the other program.

So, the 2 key advantages of web services are;

- Platform/Language independent – Because from the users point of view all that is exchanged is XML which is a text based data format. The machinery that creates this XML behind the scenes can be on any platform using any language. The ubiquity of the HTTP protocol makes it the ideal choice for integration.
- Firewall Friendly – Because they work over HTTP web service traffic travels through port 80 on any PC machine. This means that a System Admin does not have any additional firewall management headaches with extra ports being left open on a machine.

These advantages mean that any machine that has HTTP access to the server and utilises technology that can read XML can access the Web Service and access its methods.

WEB SERVICE DESCRIPTION LANGUAGE (WSDL)

Web services are defined using Web Service Description Language (WSDL). A WSDL document is essential an XML file that describes each web service methods name, parameters, return type and call conventions (GET, POST and SOAP). The WSDL is effectively the *contract* between the client and the web service that states what is expected of both the client and the service.

When designing a web service there are 2 main approaches that the developer could take depending on the development environment.

1) If using .NET and Microsoft Visual Studio (VS) it is possible to follow a 'code first' approach. Using this approach the web service methods and all the web service machinery is developed before the interface is defined (in the WSDL doc). VS will then automatically generate the WSDL file for you.
2) The second approach is to define the interface first by creating the WSDL in a 'WSDL first' approach. This approach means that you are crafting the WSDL by hand. This would mean you would need to understand the syntax of WSDL documents quite well. This method is typically used in complex cases for interoperability. VS comes to the rescue again though as it comes with a command line tool that can generate all the interface descriptions for you.

When developing an application to interact with a particular web service the system architect should consult the WSDL document of the web service to determine how to format the XML message to pass from your application.

Explanation of the WSDL elements is beyond the scope of this paper but an example WSDL document is shown below.

```
<?xml version="1.0"?>
<definitions name="Temperature"
      targetNamespace="http://www.test.com/temperature.wsdl"
      xmlns:tns="http://www.test.com/temperature.wsdl"
      xmlns:xsd1="http://www.test.com/temperature.xsd"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
      <schema targetNamespace="http://www.test.com/temperature.xsd"
            xmlns="http://www.w3.org/2001/XMLSchema">
         <element name="TemperatureRequest">
            <complexType>
               <all>
                  <element name="zipCode" type="int"/>
               </all>
            </complexType>
         </element>
         <element name="TemperatureResponse">
            <complexType>
               <all>
                  <element name="temperature" type="int"/>
               </all>
            </complexType>
         </element>
      </schema>
   </types>

   <message name="GetTemperatureInput">
      <part name="body" element="xsd1:TemperatureRequest"/>
   </message>
   <message name="GetTemperatureOutput">
      <part name="body" element="xsd1:TemperatureResponse"/>
   </message>

   <portType name="TemperaturePortType">
      <operation name="GetTemperature">
         <input message="tns:GetTemperatureInput"/>
         <output message="tns:GetTemperatureOutput"/>
      </operation>
   </portType>

   <binding name="TempSoapBinding" type="tns:TemperaturePortType">
      <soap:binding style="document"
                    transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetTemperature">
         <soap:operation soapAction="http://www.test.com/GetTemperature"/>
         <input>
            <soap:body use="literal"
                  namespace="http://www.test.com/temperature.xsd"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </input>
```

```
         <output>
            <soap:body use="literal"
                  namespace="http://www.test.com/temperature.xsd"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </output>
      </operation>
   </binding>

   <service name="TemperatureService">
      <documentation>Get your current temperature</documentation>
      <port name="TemperaturePort" binding="tns:TempSoapBinding">
         <soap:address location="http://www.test.com/temperature.temp.asmx"/>
      </port>
   </service>
</definitions>
```

SOAP

SOAP (formerly Simple Object Access Protocol) is a protocol that acts as message format that wraps the XML provided from a web service ready for transport across the wire. In the case of web services SOAP uses HTTP to transport the XML but SMTP (an email protocol) can be used as well.

Below is an example of a SOAP request taken from Wikipedia. The request example is a SOAP message to request information from a warehouse web service by providing a product ID.

```
<soap: Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <getProductDetails xmlns="http//warehouse.example.com/ws">
            <productID>8273648</productID>
        </getProductDetails>
    </soap:Body>
</soap:Envelope>
```

As you can see SOAP wraps the request inside an XML 'envelope' ready for transport.

USES FOR WEB SERVICES

The core use of web services is to provide access to remote processing power to a client user. For the most part this means that the web service acts as an interface or a façade to an underlying database. By using web services in this way, rather than allowing direct access to the database you are effectively de-coupling the client application from the database which means if you want to change that MS Access database to SAS or ORACLE you can. The client application need never know! However the advantages of web services over other remoting technologies allow them to have a couple of other major applications.

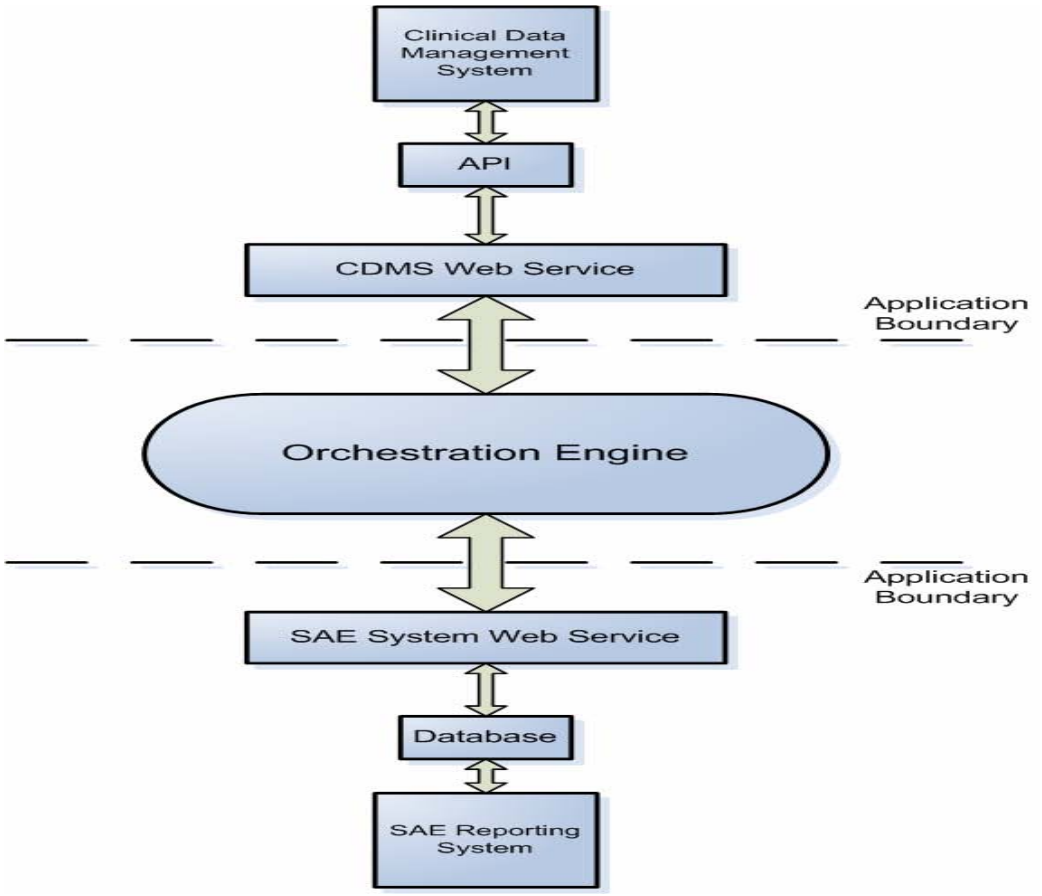
INTEGRATION OF HETEROGENEOUS SYSTEMS

There are many systems in all sorts of different businesses that do not talk to each other directly. This is sometimes not a problem if the systems are totally unrelated but often when they are it can lead to a number of problems.

1) Data can be duplicated in a number of different repositories
2) Systems 'get out of sync' with each other with data in one system being out of date compared to another system

At best these systems can share information through a semi-automated export, transformation and import process using simple files like CSV files. At worst the system may cross company boundaries and may require a data entry clerk to enter data from a data listing. Wouldn't it be nice to stop this problem!

Well web services come to the rescue. The diagram below shows how web services act as a *façade* for each of the applications. The example shows the interaction of a Clinical Data Management System (CDMS) which may be based on UNIX with an ORACLE database and a SAE Reporting System using SAS as its database engine. A common task is for a Data Manager to manually reconcile the adverse events reported in the SAE Reporting system with the adverse events recorded on the study CRFs and entered into the CDMS.



The web services assigned to each system interacts with the systems API or other communication pathway (e.g. direct database access) to create business entities that can be serialised as XML. The web service can now act as the *interface* to the application allowing clients that subscribe to its WSDL contract to interact and share data with it. Each web service can then be orchestrated through an engine such as the Windows Workflow Foundation (WWF). The WWF orchestrates XML messages and ensures that through a defined workflow data is passed from system to system keeping them all current.

BUSINESS TO BUSINESS (B2B) COMMUNICATION

Most business to business interaction tends to be via paper based systems, a typical example being invoices. The reason for this is the low cost of entry and usage of such systems. Common barriers to computerised interactions have in the past been based on the same reasons for intra-company system interaction; heterogeneous platforms and technologies. As previously outlined the advantages that web services bring are platform and technology independence. This means that as companies become more integrated in a bid to integrate supply chains or services web services can be used to fill this technology gap.

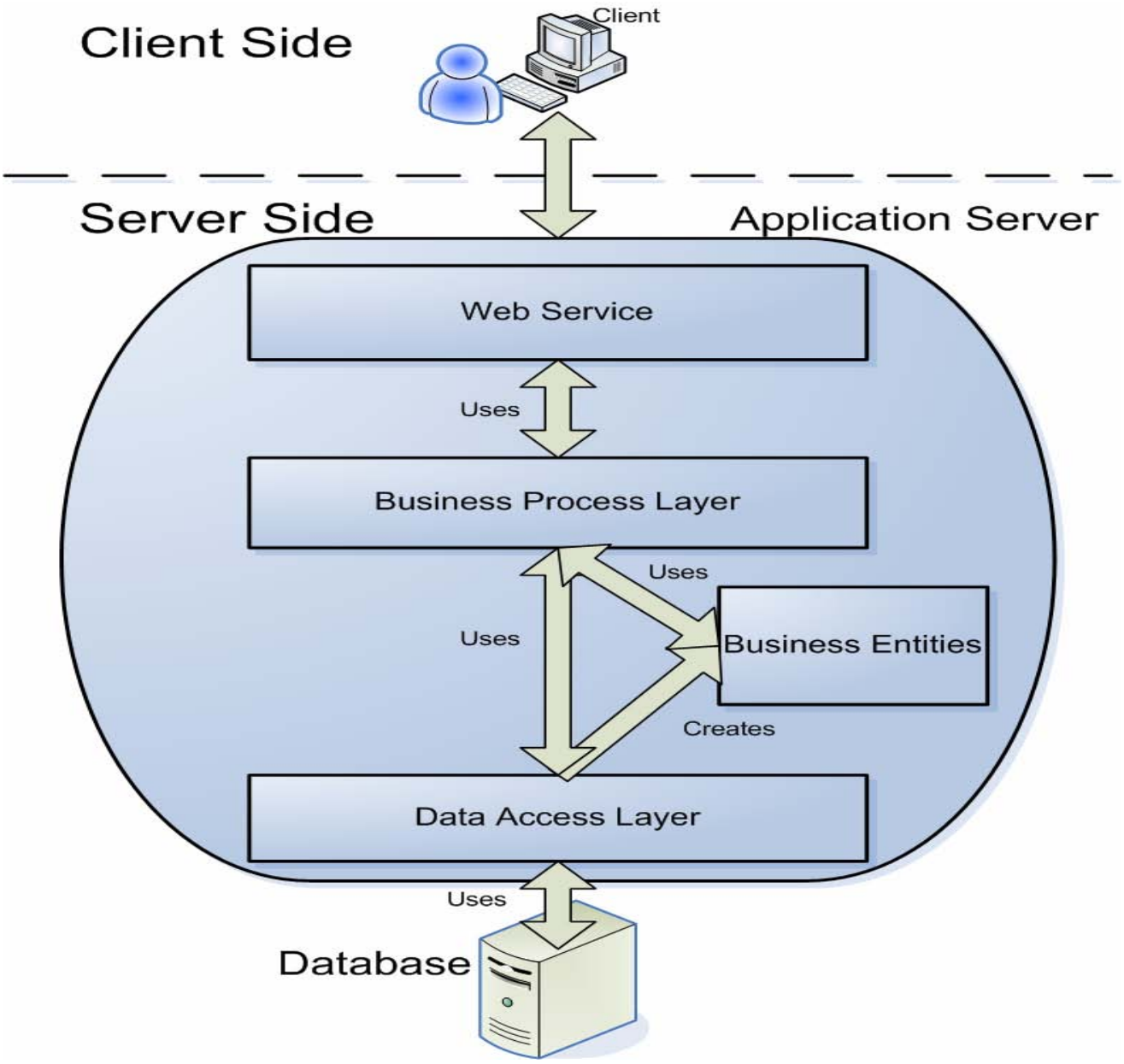
By companies designing web services around the WSDL document of a supplier's web service they can interact to process order information, invoices even project metrics (depending how far you want to go). In a pharmaceutical setting it is a goal of CDISC to make their ODM and LAB XML schemas the format for clinical data transfer.

Often clinical data is transferred between sites in a less than integrated manner using various file formats. These file formats include; raw SAS datasets, SAS transport files or even CSV files. These are typically shared via email, document management systems, FTP sites or even couriered CDs or DVDs. All these mechanisms arise because of the lack of integration of electronic communication channels between pharmaceutical companies. This area of the business is a key area in which web services could be applied. Using the publicly available ODM and LAB schemas from CDISC it would be possible for any pharmaceutical company to transfer data between their partners (CROs etc) by creating a set of secured web services. For example an external statistics department based within a CRO could use a pharmaceutical company's web service to transfer raw data from their CDMS system for analysis. External technology vendors could synchronise data from within their own product (EDC systems, patient diaries etc) with the sponsor's in-house system. Pharmaceutical companies could transfer data to the regulatory authorities (true e-submission).

With web services this is all possible even if companies use different operating systems (UNIX, Windows etc) or technologies (.NET, Java etc).

TYPICAL WEB SERVICE ARCHITECTURE

There are a number of ways that you can design the architecture of your Web services but the architecture outlined here is typical of an *n*-tier setup. The diagram below outlines our typical architecture.



Web services are typically hosted within a web server such as Internet Information Services (IIS) which Microsoft's enterprise level web server. In keeping with the n-tier architecture the processing is split into various layers. The web service is the interface that is publicly available for interaction with a web service client. All the other layers are private and therefore are not exposed to any external client. Each layer is arranged so that the only know about the layers immediately above and below. For example the Business Process Layer does not know about the client and the Web Service does not know about the data access layer. This is called 'separation of concerns' and means that each area of processing is a de-coupled from the other areas as possible. A de-coupled system is easier to extend and change.

When a client makes a request to the web service the web service calls one or more methods in the Business Process Layer (BPL). The BPL is responsible for managing and processing the systems Business Entities (BE). BE are classes and objects that represent the elements that make up the domain e.g. an employee class, a project class, a CRF class etc. In order to create the BE the BPL needs to interact with the Data Access Layer (DAL). The DAL acts as a façade to the actual database and creates the BE that the BPL will process.

Having a façade layer such as this means that the database structure or even technology can change and no other layer has to know other than the DAL.

When the BPL receives the BE from the DAL (still with me!) it process the BE and serves them back to the web service layer. This layer can then serialise this BE objects as XML ready to pass back the client.

WEB SERVICES AND .NET

The Microsoft .NET Framework provides a number of namespaces that contain a host of classes that can be used to create web services and the program logic contained within them. Using Microsoft's primary IDE (Integrated Development Environment), Visual Studio, creation is made even easier because there is a project template for creating web services. In addition to this the IDE also creates a number of the web service elements for you (if you wish). For example you can create a web service by coding it first (as opposed to creating the contract – i.e. the WSDL – first) and visual studio will create the WSDL file automatically for you.

When creating web services with .NET you write the code in one of the .NET supported languages such as C#. You write the methods for the service as you would any other within a .NET class but in order to expose your method publicly you need to 'decorate' your methods with the '[WebMethod]' attribute. This tells the compiler that this method is to be exposed as part of the web service and therefore should add it to the WSDL file.

For example

```
[WebMethod]
public string SampleMethod()
{
    return "Hello World";
}
```

When you deploy your web service to a web server (for example IIS) you can then access the methods exposed by the web service by submitting SOAP request to the web service URL.

WEB SERVICES AND SAS

So where does SAS fit into this. Well the good news is that it is possible to harness the power of SAS within our web services.

There are 2 main methods for doing this:

1. Simple SAS interaction – The developer can access SAS data by using one of the SAS data providers. For example using the SAS provider for SAS/SHARE a developer can access data stored on a SAS/SHARE server by using ADO. I call this a simple interaction because the main processing of the data is done within the technology used by the developer (e.g. .NET) rather than by SAS.
2. By using SAS Integration Technologies. This is a more advanced option because Integration Technologies allow the developer to start and manipulate SAS sessions, set up SAS libraries, submit code statements and stored procedures. This gives us the means to do more of the data processing within the SAS environment.

The objects and classes that make up the Integration Technologies (IT) package are available as part of the BASE SAS installation but in order to be able to use the full functionality you need the full IT license.

The main controller class of IT is the WorkspaceManager. This class allows you to create Workspace objects that are equivalent to SAS interactive or BATCH sessions. Once you have instantiated a Workspace object you can do all the things that you can with a normal SAS session (e.g. set libraries, submit code).

The following C# method uses the WorkspaceManager to create a Workspace object. This object is returned by then method and can be used like a normal SAS session

```
private SAS.Workspace InitialiseSAS()
{
    SASWorkspaceManager.WorkspaceManager wsm= new SASWorkspaceManager.WorkspaceManager();
    string xml = "";

    SAS.Workspace SASws = (SAS.Workspace)wsm.Workspaces.CreateWorkspaceByServer("",
SASWorkspaceManager.Visibility.VisibilityProcess, null, "", "", out xml);

    return SASws;
```

```
    }
```

This method uses the Workspace and the OleDbDataAdapter to return a .NET dataset based on the recordset returned by the SAS SQL statement.

```
    private DataSet GetSASData(SAS.Workspace workspace, string datasetName)
    {
        DataSet returnData = new DataSet();
        OleDbDataAdapter dataAdapter = new OleDbDataAdapter("select * from " + datasetName ,
"provider=sas.iomprovider.1; SAS Workspace ID=" + workspace.UniqueIdentifier);

        dataAdapter.Fill(returnData, "SASData");
        return returnData;
    }
```

By using these and other classes available to us though IT we can hook our web services to the power of SAS. Perhaps the most useful and scalable option is to use web services to execute SAS stored procedures. Stored procedures in SAS are essentially just SAS programs. Their power comes from the fact that we can dynamically substitute parameters (macro variables) into the program at run time making them more generic. A very simple example of a SAS stored procedure is shown below. The macro variables at the top of the program (above the *ProcessBody; line) can be substituted dynamically when we call and execute the stored procedure.

```
%LET indata=SASHELP.CLASS;
%LET outdata=WORK.TEMP;

*ProcessBody;

proc means data=&indata;
    output out=&outdata;
run;
```

We can create a web service method that executes SAS. In order to execute stored procedures we need to tell the workspace where the stored procedures are located.

```
workspace.LanguageService.StoredProcessService.Repository = @"file:D:\PhUSE
Presentations\PhUSE 2006 (Dublin)\StoredProcs ";
```

We can then create the parameters we want to pass into the macro. This is essentially a string of value pairs that are translated into macro variables that are placed at the top of the stored procedure before it executes.

```
string parameters = "indata=" + inputDSN + " outdata=WORK.RESULT";
```

Finally we can execute the stored procedure.

```
workspace.LanguageService.StoredProcessService.Execute(storedProcedureName, parameters);
```

CONCLUSION

Web services are the foundation of what is known as service orientated architecture. Systems developed using this architectural pattern interact with each other through exposed methods of services. HTTP is the protocol of the internet and so this makes web services the ideal choice for business-to-business (B2B) communication. The textural nature of XML is a further advantage because firewalls prefer this to binary content. Vast improvements can be made in B2B communication thought the usage of web services. Client systems can exchange information seamlessly allowing for better process integration between companies.

The methods of a web service are described by a WSDL document. A WSDL document is an XML contract that describes all the elements that make up the web service – a sort of metadata document. Clients of the web service must implement the WSDL contract in order to interact with them. Once implemented a client application can make SOAP requests to the web service. These requests instigate remote processing on the web server.

SAS can be integrated into web services using 2 methods. The first is a simple interaction with SAS that treats it as if it were purely a database product. Using this methodology the application accesses SAS data using one of the SAS data providers. For example, the SAS data provider of SAS/SHARE. All the processing work is done using the calling technology. A more advanced use of SAS it to use Integration Technologies (IT). This is a SAS product that gives you access to a number of

classes that allow you to manipulate SAS as if you were running an interactive session. Using IT you can set up SAS libraries, execute SAS code and can also run SAS stored procedures in order to harness the power of SAS in manipulating data.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Edward Foster
Oxford Pharmaceutical Sciences
Lancaster House
Kingston Business Park
Kingston Bagpuize
Oxford / OX13 5FE
Work Phone: +44 (0) 1462 477948
Email: edward.foster@ops-web.com
Web: www.ops-web.com