An Introduction to Web Services Enabled with PHP

By D. Britton Johnston, NuSphere Corporation – January 2002

Table of Contents

Web services are an evolution of the standards and protocols used to create the Web as it is known today. Web service infrastructure built on open XML, SOAP, WSDL, and UDDI standards will become as widely implemented as HTML. The implementations will be prevalent and portable, covering every operating system and programming language.

Leveraging a Web service architecture will enable the enterprise to collaborate around distributed processes with external business partners. Integration efforts, whether *within* an organization or *between* organizations, are eased, as programs written in different languages on different platforms may communicate with each other in a standards-based way. This will provide the enterprise with increased flexibility, new business opportunities and reduced integration efforts.

This paper will explore the evolution of and potential uses for Web services. This paper will also illustrate that using Web services is a straightforward evolution of Web programming models and demonstrate how PHP can be used as a fast and easy development tool for creating them.

Understanding Web Services

Web services are modular applications or functions, which are generally independent and self-describing, that can be discovered and called across the Internet or an enterprise intranet. The Web service itself may contain application logic and data. Leveraging standard Internet protocols, applications can gain programmatic access to business functions supported by the Web service. Such Web service functions may range from simple character validation to complicated algorithmic computations or business processes.

Web services are actually an evolution of the standards and protocols used to create the Web as we know it today. The HTML standard that displays content has evolved into a more generalized standard, XML, which can be used to describe and encode virtually any data. The HTTPS protocol has been extended with SOAP, which provides for a more generalized communication between two independent systems. Web Services Description Language (WSDL) enables a program to understand how it might interact with a Web service – it describes the methods and attributes associated with a Web service. Web services can be published in a directory to be found or discovered and used by other programs using a new specification that has evolved from LDAP, Universal Description, Discovery, and Integration (UDDI.)

Web services are built on XML, SOAP, WSDL, and UDDI specifications. These standards will become as widely implemented as HTML is today. The implementations will be prevalent and portable, covering every operating system and programming language. These new and improved standards provide the foundation for application integration and aggregation that companies are starting to use to build real solutions and improve the usefulness and interoperability of their applications. No longer is application functionality locked behind a veneer of HTML applied to a legacy application – Web services will allow specific business logic to be exposed and used between independent applications with a minimum knowledge of the Web service and underlying application.

Leveraging open standards, Web services are the fundamental building blocks in the evolution to pervasive, distributed computing on the Internet. After all, open standards and the emphasis

on communication and collaboration were the very accelerants that gave rise to the Web itself. These enduring trends have created an environment where Web services will continue evolving to be the platform for application integration.

Constructing applications leveraging Web services can be straightforward. Applications can be constructed using multiple Web services from varying sources that work together regardless of where they reside or how they were implemented. The functionality provided by a Web service can vary widely from the simplest character transformation or information retrieval to the most complex business operation. Basic Web services may be invoked to convert a temperature from Celsius to Fahrenheit or to retrieve a phone number or zip code. More complex Web services may be called to manage complicated transactions in an enterprise supply chain application. Regardless of the complexity of the application, Web services can be programmatically and consistently leveraged to provide functionality.

As developers build applications and the business requirements for those applications become more demanding, certain key issues must be resolved, such as reliable routing, delivery, and security, that are not yet addressed by the current basic set of standards. And while many vendors are jockeying to provide proprietary solutions to these challenges, developers should take great care to avoid being locked into one approach, platform and operating system. Only an open standards approach will ensure broad Web services interoperability and provide greater long-term value.

Using Web Services

Enterprises face many decisions regarding the use and implementation of Web services. Web services may be leveraged to create internal integration efficiencies or to extend applications to business partners or third parties. The motivation for using Web services may be in managing costs for the bottom line or in creating new business opportunities and growing the top line. Framing these options into various implementation models should help to organize decisions, opportunities, and considerations. As will be discussed, the level of desired external integration becomes the main driver of choice.

Web service implementations can be organized into the following three basic models:

1. Level I: Enterprise Application Integration
2. Level II: Single Partner Integration
3. Level III: Multiple Partner Integration

Level I: Enterprise Application Integration

Enterprise Application Integration is the use of Web services *within* a single organization. Leveraging Web services architecture, the firm may develop libraries of reusable programming building blocks to speed application development. Such Web services may be components of company-specific program logic or commonly used application functions, such as currency conversions or date calculation.

To illustrate this model, consider a hotel chain that is developing a new guest scheduling system for hotel managers. Scheduling a guest stay would not only require booking the rooms, but also require interfacing with the hotel's legacy customer management system for guest preferences and member rewards information. Leveraging a Web services architecture, the hotel chain could create an interface to the customer

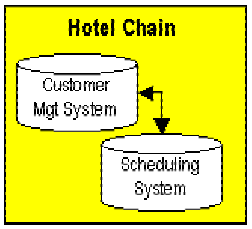


Figure 1 - Enterprise Application Integration

management system and expose it as a Web service, as depicted in Figure 1. This would allow programs written in different languages on different platforms to communicate with each other in a standards-based way, easing integration efforts. Allowing integration between applications within a single company such as this is typically the first use of Web services for large organizations, as it allows the greatest control over security and privacy.

Level II: Single Partner Integration

The next level of Web services implementation is Single Partner Integration, which extends application integration reach beyond the enterprise. Suppose the aforementioned hotel chain now wishes to gain access to a transportation partner's scheduling system in order to enable hotel managers to schedule shuttle services for hotel guests. Non-service based implementation options would require complex, lengthy, and costly integration programming. Alternatively, hotel managers could be forced to use multiple applications to complete the process of scheduling guests, "toggling" between hotel screens and transportation partner screens to perform specific tasks.

Leveraging a Web services architecture however, the transportation partner could create an interface to their scheduling system and expose it as a Web service to the hotel chain scheduling system. The hotel manager would now have easy, integrated access to transportation shuttle schedules. And because the system interfaces rely on Web services, rather than on direct integration, the hotel chain could easily swap transportation vendors or offer additional choices for hotel guests. Not only are costly and lengthy system integration efforts reduced and disruptive business processes avoided, but also greater business flexibility is provided to the hotel chain.

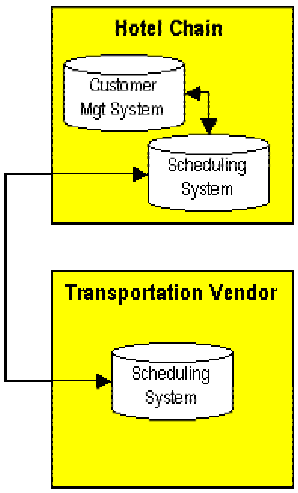


Figure 2 - Single Partner Integration

As depicted in Figure 2, Web services are shared *between* organizations that likely have formal partnerships. Components of core business applications are exposed as Web services and shared, which facilitates inter-organizational collaboration. Leveraging this architecture, applications may be constructed using multiple Web services,

from various sources, which work together regardless of where they actually reside or how they were implemented.

Level III: Multiple Partner Integration

The third model of Web service implementation is Multiple Partner Integration and is an advanced evolution of the previous model and requires the most complex levels of application collaboration. Application integration is extended to and coordinated with *multiple* business partners. This may entail integrating simple information sources, such as weather forecasts, sports scores, horoscopes, or stock quotes, or complex, critical capabilities, such as credit card verification or user authentication services. The Web services themselves may be exposed between trusted business partners or discovered in service directories.

As depicted in Figure 3, the Multiple Partner Integration model requires collaboration between multiple entities. Neither the business relationships nor the technical integration need be permanent. Infrastructure that can be readily exposed to partners and customers and supports highly distributed business processes.

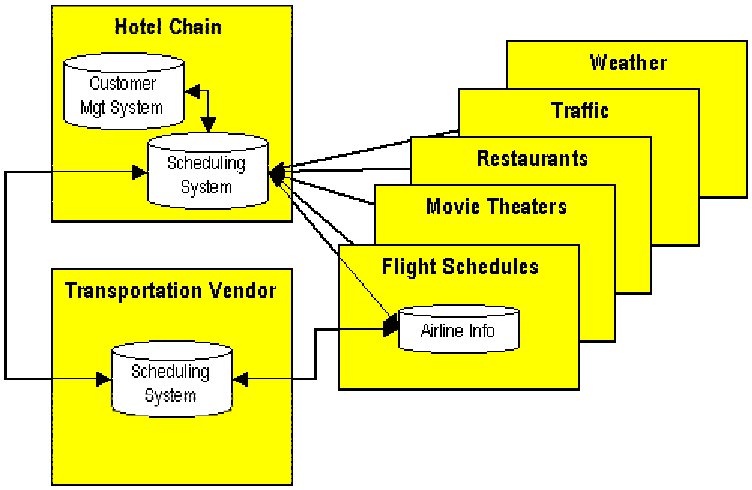


Figure 3 - Multiple Partner Integration

Building upon the hotel chain example, suppose the hotel manager wished to improve the room scheduling customer experience by suggesting and booking sightseeing activities based on

weather forecasts or traffic reports. The hotel scheduling system could have information and capabilities rolled up in an integrated and unified application for users, even though external business partners managed the underlying systems and specifications. The hotel chain may even wish to expose their scheduling functionality directly to end customers. Integrating these distributed applications and processes by "traditional" means would be prohibitively costly. And if ever completed, the hotel chain would likely be unwilling to substitute or replace vendors after working so diligently to implement each one.

While the implementation of this model will require resolution of key issues not yet addressed by current, basic standards, such as reliable routing, delivery, and security, it should illustrate the power of Web services collaboration. In the race to support Web services technically, many vendors are jockeying to establish industry standards. However, truly ubiquitous collaboration will undoubtedly require broad Web services interoperability, as envisioned above. Developers should take great care to avoid being locked into one approach with proprietary vendors, as the ultimate collaborative value of their Web services will be limited.

A Sample Web Services Application

The section of the paper will review a basic Web services application, named Contact, and present an option for approaching the development process. This will serve as an introduction to Web services, but should demonstrate the ease with which they may be created with existing technology tools, such as NuSphere PHPEd. Specific topics covered will include setting up the application, understanding the application design and underlying database structure, and implementing the client side of the Web service. While Contact Web service provides a interface to generalized database content, it could easily be extended to cover much more complex data. For the purposes of our discussion we want to minimize the complexity of the application to make it easy for anyone to use this material to understand how easy it is to write and use a web service.

A subsequent paper will be released covering more advanced development topics and will leverage an enhanced Contact application. The following topics will be covered:

- Hosting a SOAP Server
- The SOAP protocol and PHP
- The role of XML in a web service
- The ins and outs of WSDL
- How to find and use web services on the web
- Web service interoperability
- Publishing your web service
- The role of directories and UDDI

It is expected that this application will be refined and adapt over time. We would like to encourage you to send us your ideas to improve it, ideas on how to explain or demonstrate the important concepts associated with Web services. We hope to take those ideas and help everyone understand Web services and their role in building an application today.

This application contains the necessary files and documentation to install the example using some of the products found in NuSphere PHPEd. Once you have installed the Contact applica-

tion we will then walk you through how to enable Web Services so that you can take advantage of this service over the Internet. Ideally, through this example you will be able to see our demonstration or build your own Web services to interact with the Contact application.

Installation & Configuration

In this section of the white paper we will be walking you through the installation and configuration process of the Contact Application. This installation process assumes you will be using NuSphere PHPEd as a starting point.

Before installing the Contact application, ensure all NuSphere servers (MySQL, Apache, Webmin, etc) are shutdown. Also ensure that your NuSphere Installation, MySQL databases, and any related documents are backed up. Once this is completed please continue on with the following steps to setup your sample application:

1. Identify where your NuSphere installation is located. This location will be referred in this document as your *InstallDirectory.*

2. To install, unarchive the Contact Application into your *InstallDirectory*/apache. This will update your *InstallDirectory*/apache/nsdocs directory by creating a folder called "ContactAppDocs". (*This document directory will contain the necessary files to run the sample application. Additionally, the .sql file, which you will be using to create the MySQL schema and sample data will be placed in the ContactAppDocs/Data directory and is called ContactDB.sql.*)

   Linux/Unix:

   ```
   cd /usr/local/nusphere/apache/nsdocs

   tar xfvz ContactAppDocs.tar.gz
   ```

   MS Windows:

   Use Winzip or similar utility, extract into

```
C:\Program Files\nusphere\apache\nsdocs
```

3. The sample application should be unpacked at this point. The schema and data will next be loaded into the sample application.

    a. There are many ways to create your sample database and load the sample contents however in this example we will be using phpMyAdmin. In order to load the ContactDB.sql file, required to load the schema and data, you will need to start your NuSphere Administration Site. For those users on Windows you can do this through your Windows or Linux Start Menu > Programs > NuSphere > *Start NuSphere Admin Website* option. Once started, select the Administration Tab from your NuSphere Administration Web Site and select phpMyAdmin.

    b. As shown in Figure 4, Create the ContactDB database by entering **ContactDB** under the option that states: *"Create new database"*.

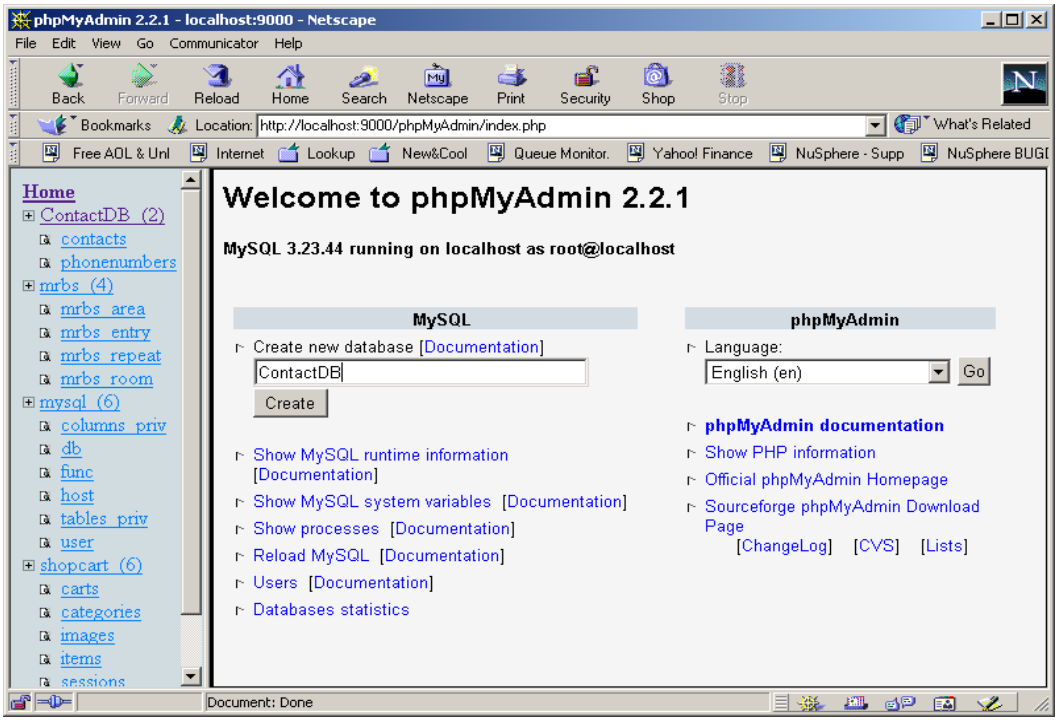


Figure 4 - Creating the ContactDB database

c. Load the schema and data by selecting to run ContactDB.sql and entering the name where it states: *"Or Location of the text file:"*, as depicted in Figure 5. The ContactDB.sql file can be found in your *Install Directory*/apache/nsdocs/ContactAppDocs/data directory. Once you have run the query for ContactDB.sql it will have loaded your schema for the ContactDB database and the sample data. You can verify this by using phpMyAdmin and viewing the schema definitions and data.

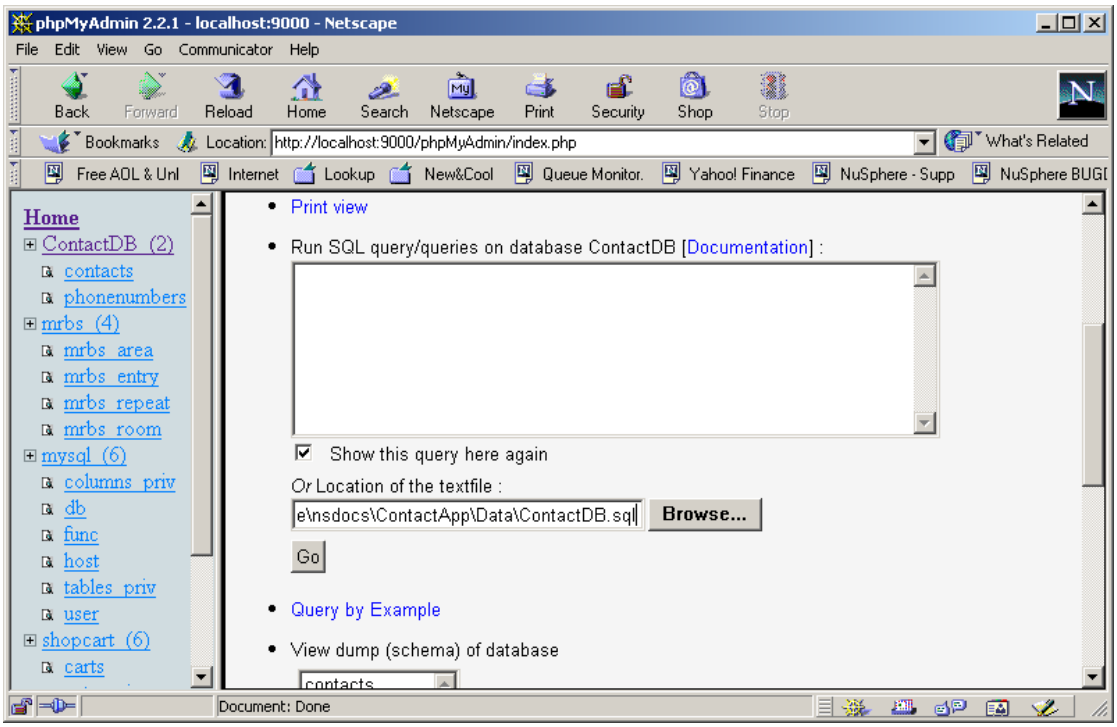


Figure 5 - Loading the schema and data

4. Now that you have successfully created the ContactDB database you may view your Contact Application. Please keep in mind that you need to make sure that your Apache server and MySQL database server should still need to be running. As shown in Figure 6, view the application by pointing a web browser to the following URL: http://localhost:9000/ContactAppDocs/Index.php.

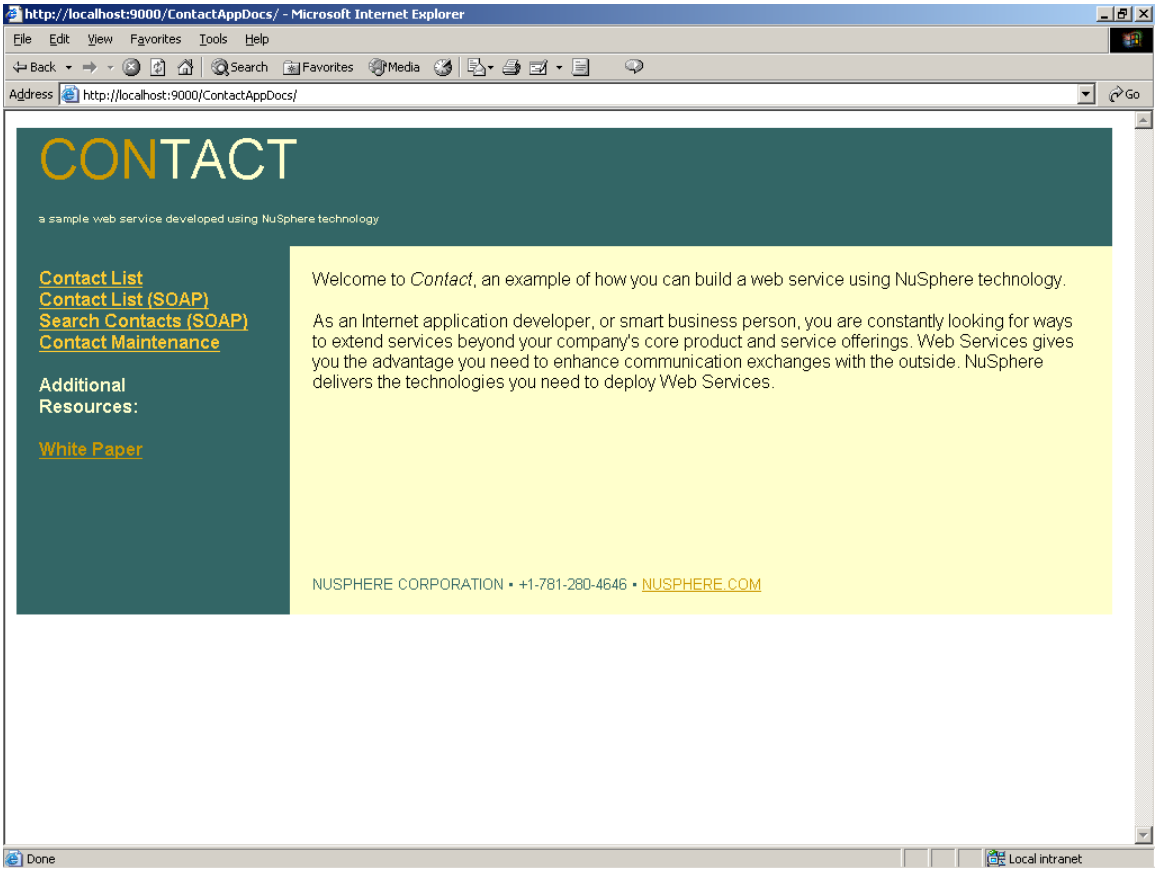


Figure 6 - The Main Contact Manager Screen

At this point you have successfully installed and configured your Contact Application. If you encounter any problems with the installation or configuration of the Contact Application please contact NuSphere Technical Support by emailing us at **support@nusphere.com**.

Application Structure

The following sections describe Contact and how it is structured. The Contact application attempts to show both traditional PHP programming and Web service approach in the same application. In this paper we will focus on the PhoneList.php and PhoneList_soap.php portions of the application.

There are four main screens in the ContactManager application. If you have everything installed and configured correctly you should be able to view all of this from your own browser.

The screen shots shown here are from a Windows system running the Internet Explorer browser, but could just as easily be from a Linux machine.

Home Page – Index.php

This script is simply a control mechanism. It primary purpose is to present to the user a menu of the available pages, and allow navigation to those pages, as shown in Figure 7.

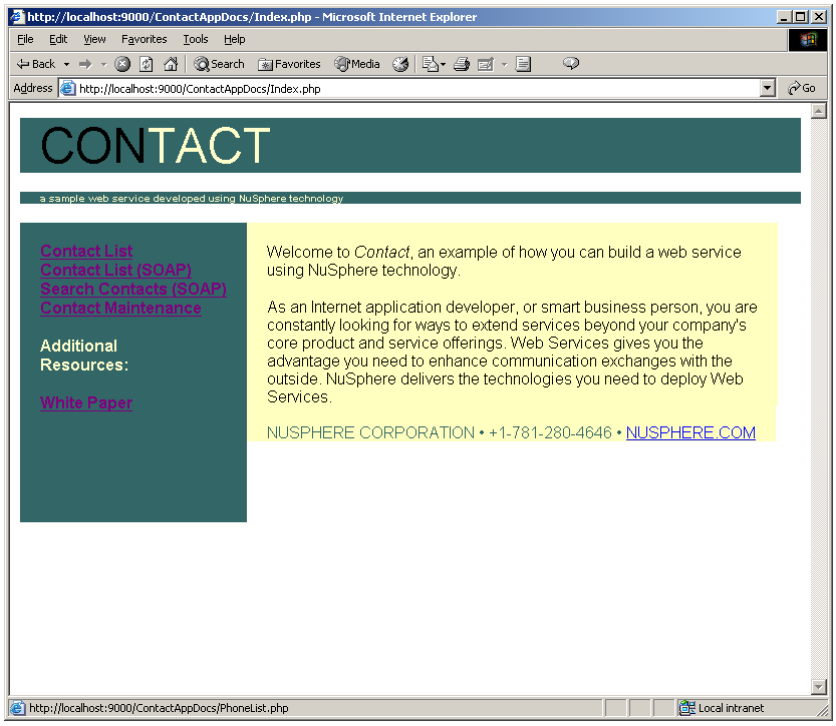


Figure 7 -- Home Page Screen

Details of the code structure follow:

Index.php	**Main index program**
→ PhoneList.php	Listing of contacts & their phone numbers
→ PhoneList_soap.php	Listing of contacts using Soap
→ Search.php	Search of the contacts using Soap
→ ContactMaint.php	Maintenance utility to update contacts & phone numbers

Contact List – PhoneList.php

This is a traditionally coded PHP script that accesses a database to extract the available contacts in the database. Shown in Figure 8, the purpose of this page is present to the user a list of contacts currently stored. Optionally, an individual contact can be selected for maintenance (refer Contact Maintenance, below).

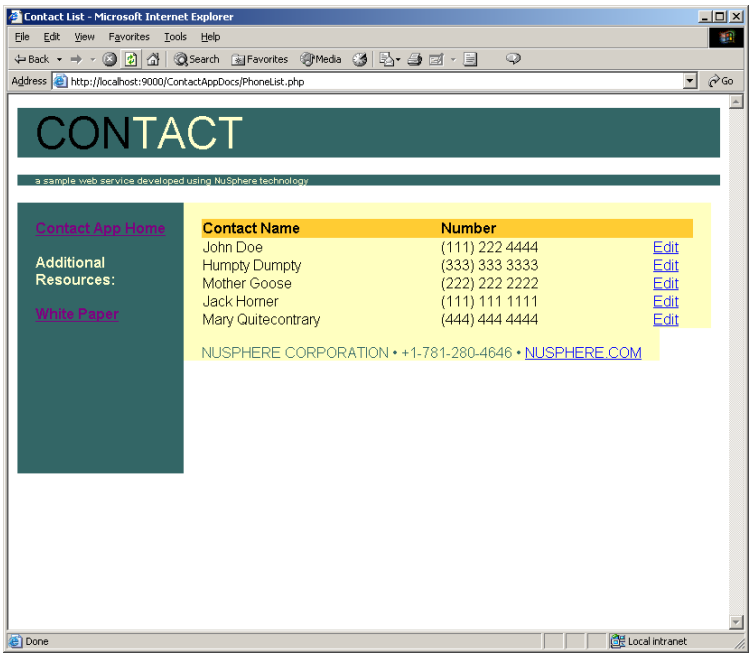


Figure 8 -- Phone List Screen

Details of the code structure follow:

PhoneList.php
- IncludeFiles/DBFunctions.inc
- IncludeFiles/PageFunctions.inc
- Index.php — Return to Home page

Contact List SOAP – PhoneList_soap.php

This is the output using a Web services architecture for the application, as shown in Figure 9. It is interesting to take a look at the debug data option, clicking on it provides a summary of the interaction between the SOAP client and SOAP server used to implement this function.

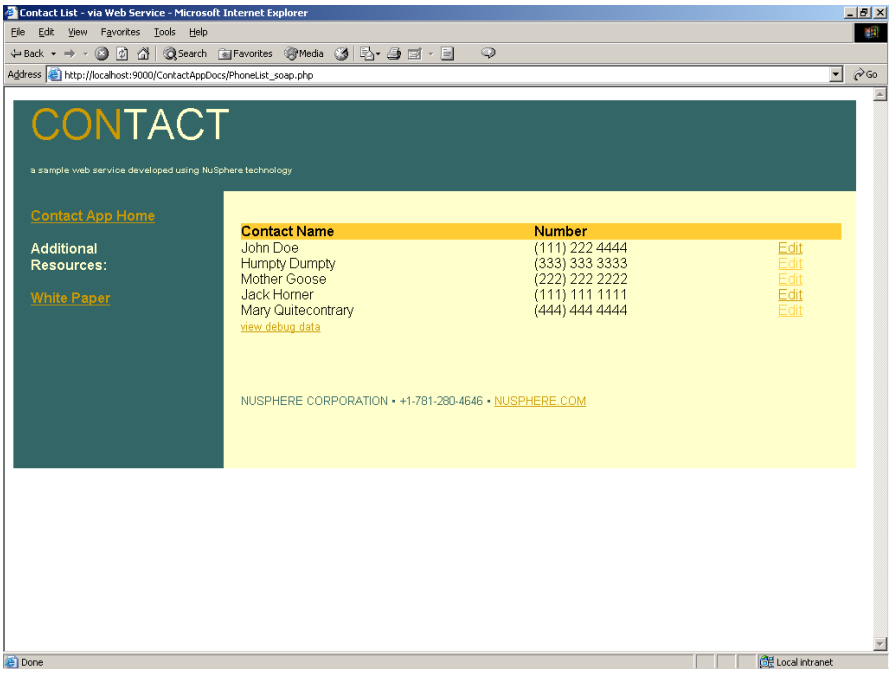


Figure 9 - Output Using a Web Services Architecture

Details of the code structure follow:

PhoneList_soap.php
- IncludeFiles/DBFunctions.inc
- IncludeFiles/PageFunctions.inc
- Index.php — Return to Home page

Search Contacts Soap – Search.php

The Search Contacts page, highlighted in Figure 10, provides a simple search mechanism; it is a more complex function than the simple list.

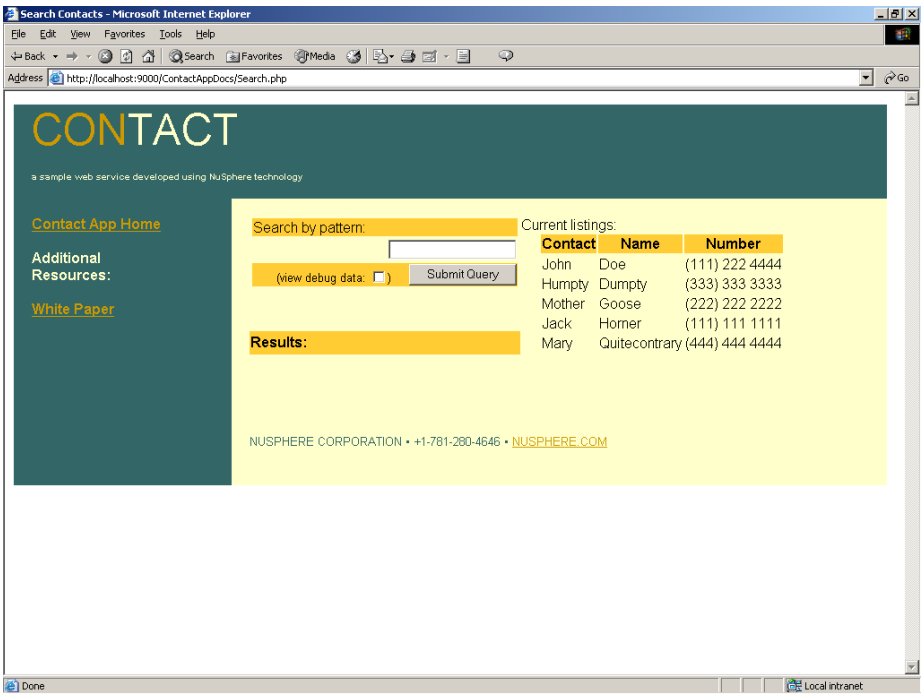


Figure 10 -- Search Contacts Screen

Details of the code structure follow:

PhoneList_soap.php
- IncludeFiles/DBFunctions.inc
- IncludeFiles/PageFunctions.inc
- Index.php — Return to Home page

Contact Maintenance – ContactMaint.php

The Contact Maintenance screen allows new contacts to be added and existing contacts to be edited. This function will not be reviewed in detail, but is referenced in Figure 11 for completeness. In our application this function is not implemented as a Web service, but as a traditional PHP application. To change an existing contact, you must enter this script by choosing the 'edit' link, next to a contact name in the Contact List screen, above.

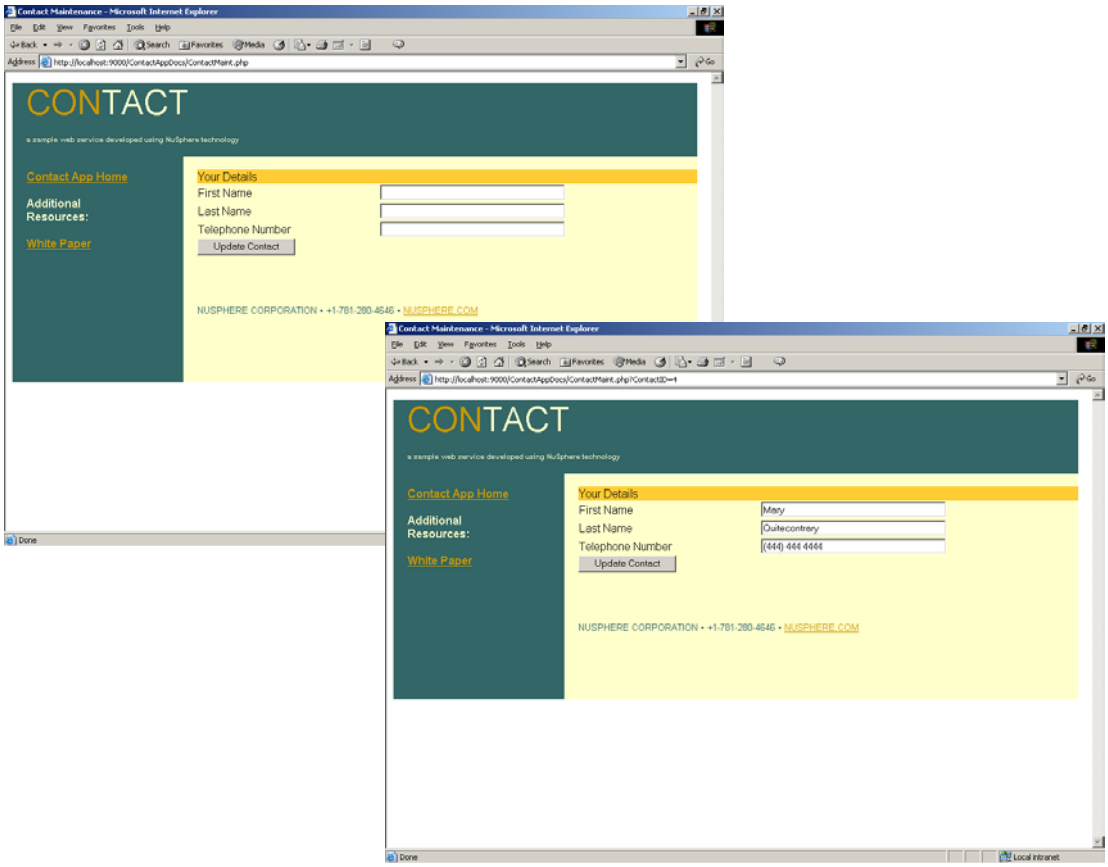
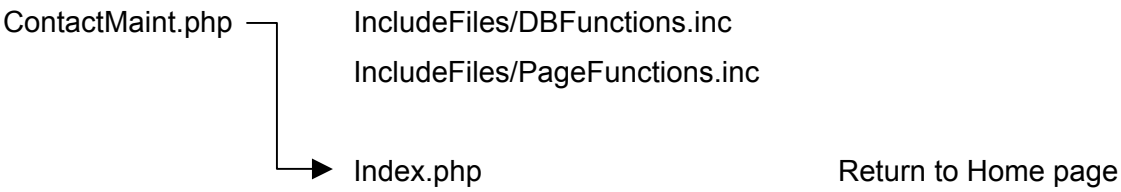


Figure 11 -- Contact Maintenance Screens

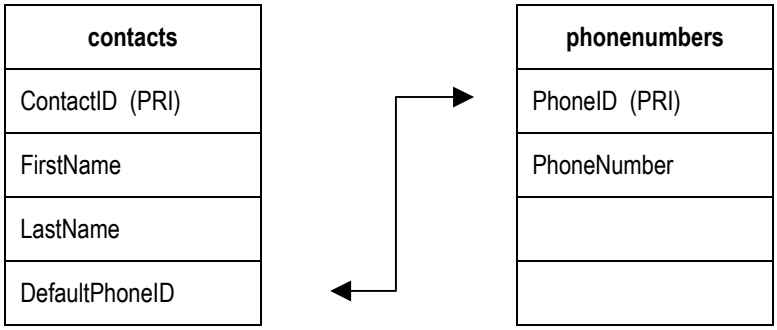
Detailed code structure follows:



Database

Before we jump into the details of programming we first should cover the basic database structure. The database is named ContactDB and consists of two simple tables. The database could have been a simple single table, but this allows us to leverage relational join operation between two tables and allows an individual to have more than one telephone number.

'ContactDB'



For the sake of simplicity, these tables have a one-to-one relationship. Sometime in the future it will/could become a one-to-many. The purpose was to show more than a trivial one-table system.

In our implementation we are using the GEMINI table type. This allows our database to enjoy transaction, row-level locking and crash recovery features provided by GEMINI.

Programming the Web Services Client

Our sample application is supplemented with two very simple applications. The programs basic.php and basicsoap.php provide a simplified view of the process required to call a Web service and we will review those here in detail.

The sample application is a little more complex. The business logic is separated from the user interface and in fact if you look at the SOAPServer.php, it completely isolates business logic. This eases maintenance and allow new function to be added to the Web service without requiring any changes in the user interface, we will take advantage of this in future papers.

The programming style is decidedly object oriented. NuSphere's PHPEd development environment directly supports object-oriented PHP coding with specific class browsing, debugging and navigation support.

Begin with the Basics

Given the Web Service is already setup, as is the case for our sample program, the process of adapting a PHP script from using a database as a data source to using a web service as a data source is simple and straightforward. The model is the same:

1. Define access infrastructure
2. Initiate the connection to the data source
3. Query the data source
4. Print the results of the query

Below are two examples of PHP scripts that print out the contents of a simple contact list. The first example shows the traditional model, which queries a database server for the information. The second example shows a web services version: It generates a SOAP message encoded in XML, sends it to a SOAP server, receives a response, and prints its contents.

Example 1: A basic PHP and MySQL script

```
<?php
//basic php and mysql script

// initiate the connection to the database
mysql_connect("localhost", "root");

// select a database
mysql_select_db("ContactDB");

// execute the query
$sql = "SELECT FirstName, LastName, PhoneNumber
        FROM   contacts, phonenumbers
        WHERE  contacts.DefaultPhoneID = phonenumbers.PhoneID
        ORDER BY contacts.LastName";
$result = mysql_query($sql);

// print the query results
while($contact = mysql_fetch_assoc($result))
{
  print $contact[FirstName]."
".$contact[LastName].",".$contact[PhoneNumber]."<br>";
}

?>
```

Example 2: A basic PHP and SOAP script

```
<?php
// basic php and soap example

// include the SOAP client classes
require_once("IncludeFiles/class.soap_client.php");

// instantiate the SOAP client object, passing it the URL of the service pro-
vider
$SOAPServerURL =
"http://localhost:9000/ContactAppDocs/Server/SOAPServer.php";
$soapclient = new soapclient($SOAPServerURL);

// invoke the 'call' method, pass method name, parameters, method namespace
$phonebook = $soapclient->call(
    "getAllListings", array(), "urn:nusphere-web-services");

// loop through results, printing each
if ($phonebook) {
    foreach($phonebook as $contact) {
        print $contact[FirstName]."
".$contact[LastName].",".$contact[PhoneNumber]."<br>";
    }
}
?>
```

Understanding the Web service

STEP 1: Include the SOAP client classes.

```
require_once("IncludeFiles/class.soap_client.php");
```

This makes the classes available to you in your script.

STEP 2: Instantiate the SOAP client object.

```
$SOAPServerURL =
"http://localhost:9000/ContactAppDocs/Server/SOAPServer.php";
$soapclient = new soapclient($SOAPServerURL);
```

The next step is to create an instance of the "soapclient" object. This object is a high-level proxy object that hides many of the details of a SOAP transaction from the user, thereby creating an easy-to-use interface to an otherwise complicated remote procedure call.

The argument that is passed to the constructor must be a URL to a valid "endpoint." An end-point, in web services jargon, is the URL at which a SOAP server resides. It is the URL to which our SOAP client will post its request.

STEP 3: Call the service

```
$phonebook = $soapclient->call(
    "getAllListings", array(), "urn:nusphere-web-services");
```

This "call" method is where the "magic" happens. The method takes 3 arguments:

1. Method name – the name of the remote service you are calling.
2. Parameters – an array of values to pass to the remote service, if required.
3. Method namespace – an optional signature for the method call.

It is within this method call that the client object generates an XML message that it posts to the SOAP server via HTTP. Upon receiving a response from the server, it parses the incoming XML message, and converts the encoded data to PHP native types, such as strings, integers, arrays and objects.

The "call" method returns mixed values. If the call transaction is successful, the return value will be an array of the value or values returned by the server. If the transaction was not successful, the method will return an array containing information regarding the reasons why it did not succeed.

STEP 4: Print the results

```
foreach($phonebook as $contact) {
        print $contact[FirstName]."
```

```
".$contact[LastName].",".$contact[PhoneNumber]."<br>";
    }
```

This step loops through the return values, printing out each.

Reviewing the Sample Programs

With the groundwork established now is the time to jump right into the source code provided in the application. Using a debugger to step through execution of basic.php and basicsoap.php is a very good next step in learning how a Web service client works. When using NuSphere® PHPEd™ and its graphical debugger it is possible to view both the client and the server side of the Web service executing allowing clear view of the interaction and the SOAP and XML under-pinnings, as depicted in Figure 12. Now is also a good time to jump into the Contact sample application. The PhoneList.php and PhoneList_soap.php provide much the same function as basic.php and basicsoap.php in the context of a Web application. The search function in Search.php uses several method from the Web service to provide searching of the Contact database on the server side. Finally, from this you should be able to get started creating your own Web services with a clear understanding the underlying technology.

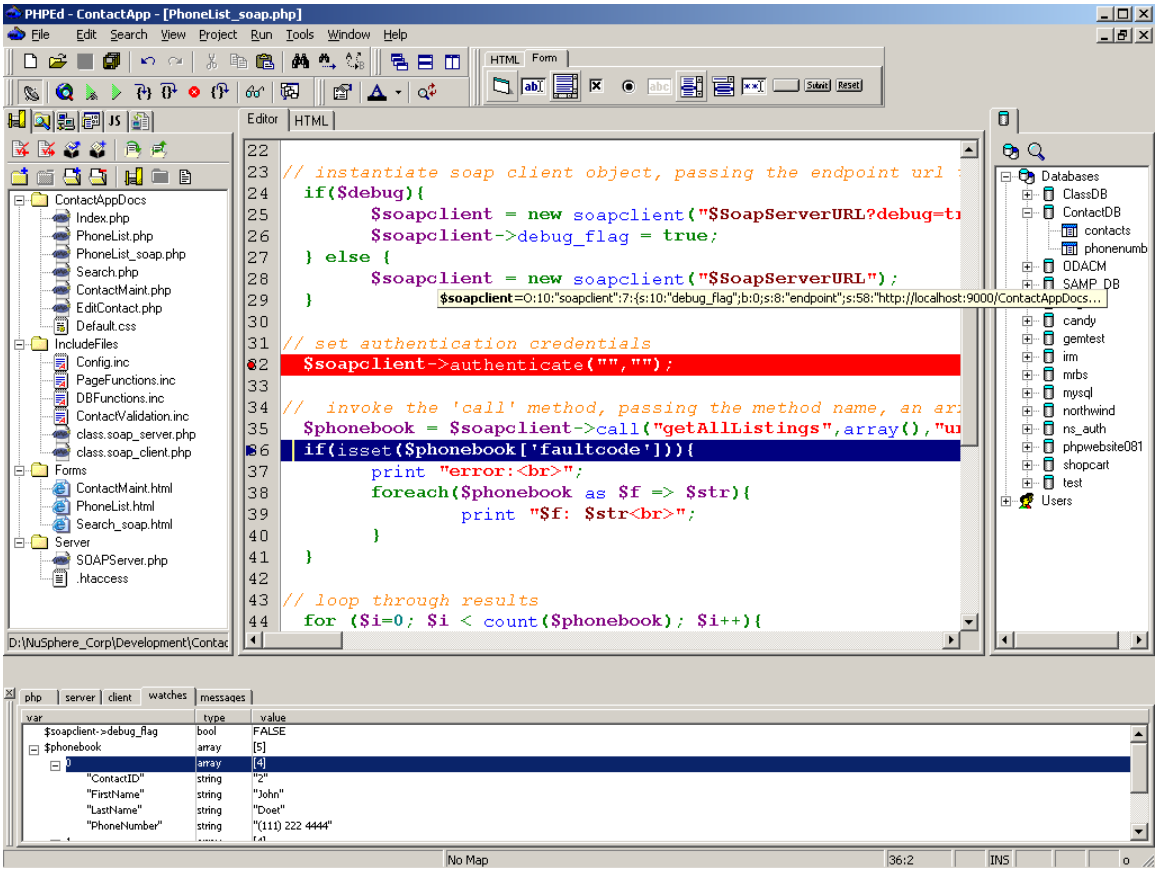


Figure 12 - Reviewing the Sample Programs with NuSphere PHPEd Editor

Summary

Using Web services is a straightforward evolution in Web programming models. And creating a Web services client is simple, as we have shown here. As evidenced above, it takes about the same amount of coding as a traditional database query, but without having to know the specific functions necessary to query the underlying database.

Web service infrastructure built on XML, SOAP, WSDL, and UDDI standards will become as widely implemented as HTML is today. The implementations will be prevalent and portable, covering every operating system and programming language. These new and improved standards provide the foundation for application integration and aggregation that companies are starting to use to build real solutions and improve the usefulness and interoperability of their applications. No longer is application functionality locked behind a veneer of HTML applied to a legacy application – Web services will allow specific business logic to be exposed and used between independent applications with a minimum knowledge of the Web service and underlying application. The magnitude of this cannot be overemphasized: disparate organizations can integrate application components and data, collaborating on distributed processes.

Truly ubiquitous collaboration will undoubtedly require broad Web services interoperability. Only an open standards approach will ensure this. In the race to support Web services technically, many vendors are jockeying to establish *their* industry standards around security, reliable routing and delivery. While these key issues must be addressed for widespread Web services adoption, enterprises should take great care to avoid proprietary approaches. Proprietary vendors will lock the developer in to an offering, which limits future migration and Web service interoperability. The implementations must be prevalent and portable, covering every operating system and programming language. Leveraging open standards will ensure that no matter what programming language is used, the Web services will not have to be re-coded to interoperate on various platforms.

About NuSphere Corporation

NuSphere delivers the first Internet Application Platform (IAP) based on open source components, providing an integrated foundation that allows companies to deploy reliable, cost-effective, enterprise-class applications across Windows, UNIX and Linux environments. NuSphere® Advantage is an integrated software suite that pairs the reliability and cost-effectiveness of PHP, Apache, Perl and open source databases with new technology for building business-critical web applications and web services. NuSphere Pro Advantage was named "Best Developer Tool" at LinuxWorld Fall 2001. Based in Bedford, Mass., the company's commercial software services include technical support, consulting and training. For more information, visit www.nusphere.com or call +1-781-280-4600.