White Paper

Introducing .NET My Services

By Markus Egger, President, EPS Software Corporation

Abstract

This white paper provides an overview on the set of consumer-focused XML Web services called .NET My Services. .NET My Services enable developers to leverage the data-centered nature of XML Web services to build user-oriented Internet applications for Web sites, Web services, applications and devices.

Specifically, this white paper provides details on how you can interact and access .NET My Services using SOAP and Visual Studio .NET.

Contents

Overview

.NET My Services is Microsoft's first attempt at professional, and widely available, commercial Web services. .NET My Services (formerly codenamed "Hailstorm") is a set of services that include everything from authentication services (.NET Passport), to storage services (.NET My Lists and others), as well as communication services such as .NET My Alerts, and even organizational services such as .NET My Calendar.

The initial set of .NET My Services will include:

- **.NET Profile.** Name, nickname, special dates, picture, address.
- **.NET Contacts.** Electronic relationships/address book.
- **.NET Locations.** Electronic and geographical location and rendezvous.
- **.NET Alerts.** Alert subscription, management, and routing.
- **.NET Presence.** Online, offline, busy, free, which device(s) to send alerts to.
- **.NET Inbox.** Inbox items like e-mail and voice mail, including existing mail systems.
- **.NET Calendar.** Time and task management.
- **.NET Documents.** Raw document storage.
- **.NET ApplicationSettings.** Application settings.
- **.NET FavoriteWebSites.** Favorite URLs and other Web identifiers.
- **.NET Wallet.** Receipts, payment instruments, coupons, and other transaction records.
- **.NET Devices.** Device settings, capabilities.
- **.NET services.** Services provided for an identity.
- **.NET Lists.** General purpose lists.
- **.NET Categories.** A way to group lists.

As of this writing, .NET My Services is still in a very early beta stage. You can download the technology preview .NET My Services SDK from http://msdn.microsoft.com/MyServices. At this location, you can obtain a user name and password to download the SDK from Microsoft's BetaPlace web site.

Note that, as with all beta versions, information provided in this white paper may (and is likely to) change by product release!

Why .NET My Services?

People frequently ask me why they would want to use or support .NET My Services. After all, many of the provided services appear to be rather simple: Calendars, Contacts, Alerts and the like. What is the great advantage of storing contact information in .NET My Services rather than Exchange or Outlook?

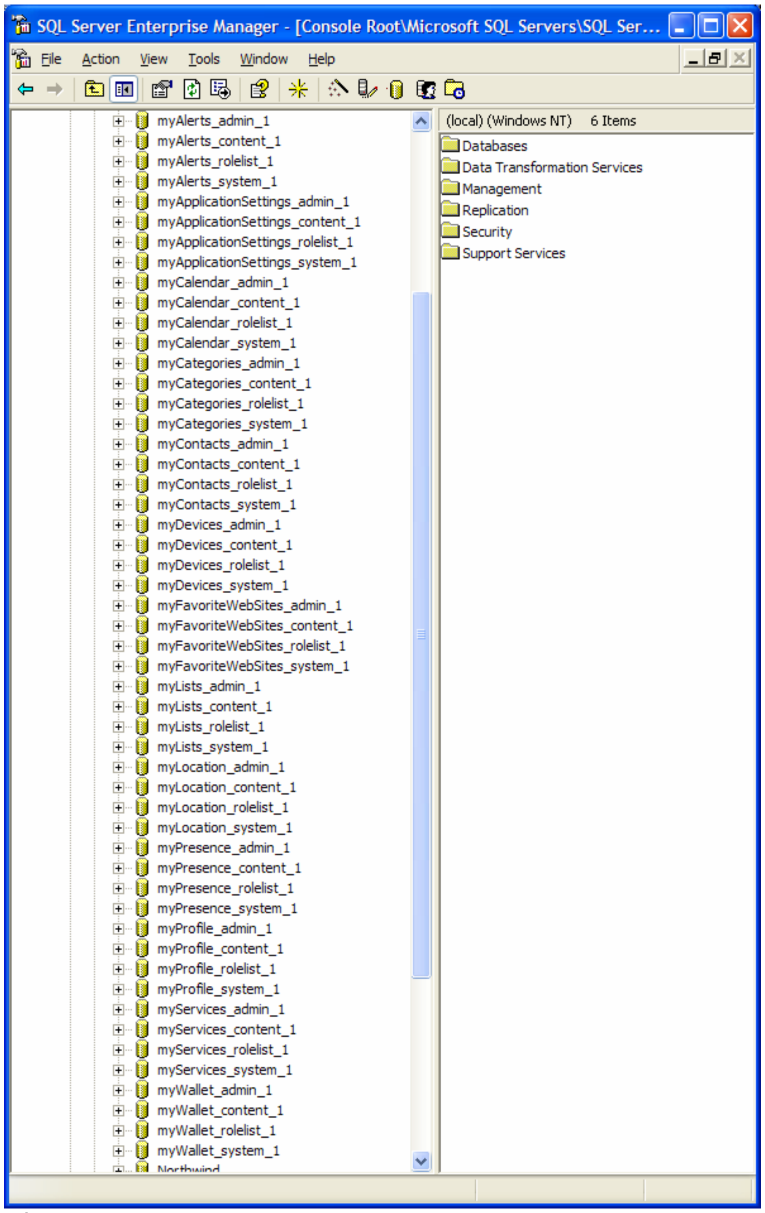
The biggest advantage is the centralized fashion of .NET My Services (I will refer to .NET My Services as NMS from here on). Contacts stored in the central repository are available anywhere, anytime, as long as the device or application you use is connected to NMS or is at least capable to synchronize with NMS from time to time. So not only can you see those contacts in Outlook or Exchange, but they can also be available on your cell phone and other devices. However, NMS doesn't stop there! Contact information is also available to Web sites and other applications you use. For instance, you can use an online travel agent and, once you book a flight, the information gets added automatically to your calendar and a notification by email is sent to specified people in your My Contacts repository.

Note that contact information tends to be outdated all the time. A friend’s address in your contacts may be wrong by the time you use it. However, let me introduce you to another service: .NET My Profile. This service stores information about you, rather than another person. You can allow other people to subscribe to that profile. This means that someone can keep an entry in his or her contacts that is synchronized with your profile, and the next time they look up your information, it will be perfectly up to date. You can use your profile for all kinds of things. If you have recently moved (as I have), you would surely appreciate if all of your bills, all of your magazine subscriptions and all your friends would automatically know of your new address.

Of course, all of this raises security issues! For that reason, the user has to give consent to all third-party interactions with your data. The online travel agent cannot just access your contacts. Instead, you have explicitly grant access, and in order to do so, you will have to identify yourself. This is handled through Kerberos security tickets (tokens). The new .NET Passport service provides that level of authentication.

Installing and Configuring the SDK

Once you download the SDK, follow the installation instructions. The initial process is straightforward. Simply choose to install the SDK as well as all the samples. You need to run the setup on the development server you would like to use. One of the preconditions is SQL Server 2000. During the setup process, the SDK installs a plethora of databases, as shown in **Figure 1**.


SQL Server Enterprise Manager - [Console Root\Microsoft SQL Servers\SQL Ser...
File Action View Tools Window Help

myAlerts_admin_1
myAlerts_content_1
myAlerts_rolelist_1
myAlerts_system_1
myApplicationSettings_admin_1
myApplicationSettings_content_1
myApplicationSettings_rolelist_1
myApplicationSettings_system_1
myCalendar_admin_1
myCalendar_content_1
myCalendar_rolelist_1
myCalendar_system_1
myCategories_admin_1
myCategories_content_1
myCategories_rolelist_1
myCategories_system_1
myContacts_admin_1
myContacts_content_1
myContacts_rolelist_1
myContacts_system_1
myDevices_admin_1
myDevices_content_1
myDevices_rolelist_1
myDevices_system_1
myFavoriteWebSites_admin_1
myFavoriteWebSites_content_1
myFavoriteWebSites_rolelist_1
myFavoriteWebSites_system_1
myLists_admin_1
myLists_content_1
myLists_rolelist_1
myLists_system_1
myLocation_admin_1
myLocation_content_1
myLocation_rolelist_1
myLocation_system_1
myPresence_admin_1
myPresence_content_1
myPresence_rolelist_1
myPresence_system_1
myProfile_admin_1
myProfile_content_1
myProfile_rolelist_1
myProfile_system_1
myServices_admin_1
myServices_content_1
myServices_rolelist_1
myServices_system_1
myWallet_admin_1
myWallet_content_1
myWallet_rolelist_1
myWallet_system_1
Northwind

(local) (Windows NT) 6 Items

Databases
Data Transformation Services
Management
Replication
Security
Support Services

Figure 1

In order for the setup program to install those databases, you have to provide a user name and password. Make sure you do NOT use the SA account without a password! Although the installation will complete without any problems, your installation will not be functional and you will receive error 0x00000012 for most everything you attempt to do.

At this point you may wonder why all those databases are installed on your server. After all, .NET My Services is a Web service framework that's supposed to run on Microsoft's servers. However, the beta SDK installs all services locally. This is referred to as ".NET My Services In-A-Box." For those who have implemented previous versions of Microsoft Passport, you will appreciate this concept. It enables you to run and develop your .NET My Services client applications locally without signing up with a test server. This used to be one of the most difficult aspects of developing Microsoft Passport sites.

After the initial setup, there are a few configuration steps left. You have to provision your server to allow for local authentication. In general, .NET My Services uses the .NET Passport service for authentication, which is required for all other services. In your local environment however, you can provision your computer to accept local user credentials instead, which greatly simplifies development.

The SDK provides command line utilities to perform the provisioning among other tasks (you can actually interact with the services using a command line utility, rather than establishing a connection to the local Web service). To use any of the command line utilities, you need to set an environment variable. To do so, open the *My Computer* Properties. On the *Advanced* tab, click **Environment Variables**. Under *System Variables*, click **New** to add a new one. Set the variable name to be SRVLOCATOR, and the value to http://server_name where "server_name" is the name of the server on which you installed the system.

Alternatively, you can also set this variable when you open a command window ("DOS window") in the following manner:

```
set SRVLOCATOR=http://server_name
```

On my computer, I use the following command, since my computer name is "megger01":

```
set SRVLOCATOR=http://megger01
```

Once you define the server variable, you can provision the system on a service-by-service basis, for each user. To do so, use **hsprov.exe**, a utility you can find in the \Microsoft .NET My Services SDK\Bin directory. To provision a service, you need to pass the name of the service as well as the user name and server name as parameters, in the following fashion:

```
hsprov.exe –s myFavoriteWebSites –l http://localhost –o markus
```

You can provision all services in this fashion, with the exception of the main service (called myServices) itself. Do NOT attempt to provision that service. If you do provision the service that way, you risk breaking your installation altogether.

Note that you will have to provision every service for every user. Note also, that the specified user names actually have to have (domain) user accounts. The documentation states that the system is provisioned for all services for the user account that performed the installation. In my experience, that is not the case. I had to manually provision all the services I attempted to use.

Interacting with .NET My Services

As the name suggests, NMS is a Web service. This suggests that the interaction with NMS is purely HTTP based. However, this is not the case. Although HTTP is a perfectly viable solution (and probably one of the most frequently used), NMS can also be accessed through WSDL and other protocols. In fact, a command line based utility called **hspost.exe** can be used for simple interaction. There is no reason for NMS to be limited to these access technologies and protocols either and there may be other ways in the future.

Interaction with NMS is based entirely on XML. You can envision the NMS database (after all, NMS is largely about data) as a big XML document against which you can run queries and commands. The commands are sent in the form of XML-based requests. The standard these requests are based on is called HSDL. Much of the syntax within HSDL is based on XPath, allowing you to query NMS data similarly to how you would query a local XML file. The HSDL-based request is usually sent within a SOAP package. The sum of all of these building blocks is referred to as XMI, which stands for XML Message Interface.

As you can see, there is no shortage of acronyms here. Luckily, most of these things are based on standards and techniques commonly used in the world of XML, so most everyone familiar with XML and the concept of Web services will feel right at home in the NMS world.

You may have noticed that a lot of things start with the letters HS: HSpost.exe, HSprov.exe, HSDL,… I would guess that HS stands for Hailstorm, which was the original NMS code name. I would expect this to change before release.

As indicated above, most of NMS is based on data, as well as the services around it. Those services as well as the infrastructure needed to run NMS is generally referred to as the NSM *Service Fabric*.

The Basic Interaction Idea

No matter what service you interact with, if will always conform to the same basic idea: A .NET Passport security ticket sends an HSDL request wrapped on SOAP to the service fabric, which accesses its data store to perform the specified action according to the schemas and rules defined by each service. An overview of this process is shown in **Figure 2**.

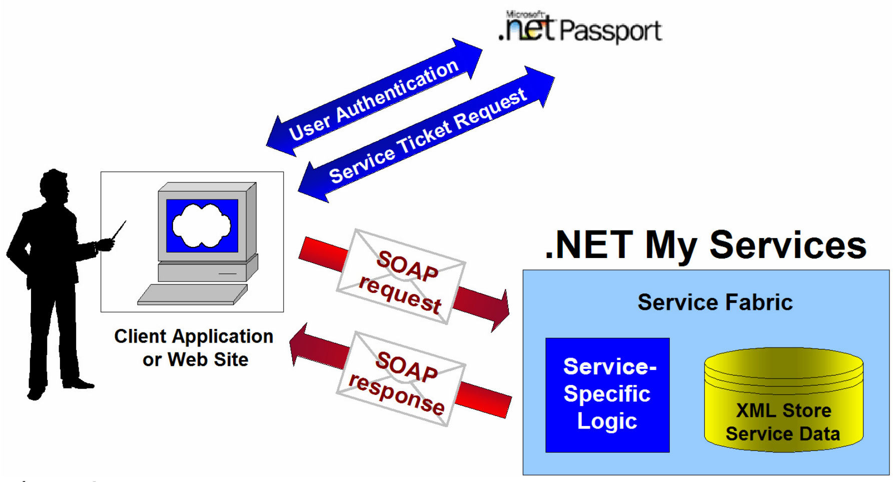


Figure 2

Understanding HSDL and XMI

Apart from security, each NMS service is made up of key XML nodes that you can query and manipulate using HSDL. The .NET My FavoriteWebSites services for instance (which can be used to store links to favorite Web sites similar to the favorites in Internet Explorer or other browsers) presents data in the following format:

```
<favoriteWebSite id="137E8DD0-DB35-4FDC-8C40-238C41A1CF92">
   <title xml:lang="en">EPS Software Corp.</title>
   <url>http://www.eps-software.com/</url>
</favoriteWebSite>
```

As you can see, this is straightforward. Each site has a unique ID (GUID) as well as a title and an associated URL. The .NET My FavoriteWebSites services defined all of this in an extensive schema. There is quite a bit more information that can be stored with favorite Web sites. The example above simply represents the minimal information required by the FavoriteWebSites schema.

When information about favorite Web sites is returned by NMS, it generally is embedded in a query response message that holds any number of favorite Web sites. Here is a typical response:

```
<hs:queryResponse xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites">
   <hs:xpQueryResponse status="success">
      <m:myFavoriteWebSites>
         <m:favoriteWebSite id="C30C45FB-11DA-4CC0-8EE6-70FDD12858D2">
            <m:title xml:lang="en">Microsoft Corp.</m:title>
            <m:url>http://www.microsoft.com/ms.htm</m:url>
         </m:favoriteWebSite>
         <m:favoriteWebSite id="137E8DD0-DB35-4FDC-8C40-238C41A1CF92">
            <m:title xml:lang="en">EPS Software Corp.</m:title>
            <m:url>http://www.eps-software.com/</m:url>
         </m:favoriteWebSite>
      </m:myFavoriteWebSites>
   </hs:xpQueryResponse>
</hs:queryResponse>
```

This looks a bit more complicated, but it really isn't. The first node is simply a wrapper for the queries sent to the service (we'll discuss the query that lead to this a bit later). It also defines the code NMS namespace as well as the namespace specific to the FavoriteWebSites service, and assigns **hs** and **m** respectively, as the namespace identifiers.

Then, we receive the first real query response wrapped in a *xpQueryResponse* tag (XPath Query Response). It is assigned the **hs** namespace. The node also indicates that the query ran successfully. For that reason, the content of the xpQueryResponse tag is NMS data. Otherwise, the content would be an error message.

The remaining XML is identical to the simple version above, except that there are two favorite Web sites that returned by this query. Also, all the tags used to hold the actual data are tied to the **m** namespace.

Of course, other NMS services present data in different formats. Each NMS service has its own schema.

Each schema defines some nodes that are more relevant than others. Some of those nodes you can retrieve directly, others can only be part of a larger result set. Some of the nodes can be used in queries, others can't. Nodes are also color-coded. NMS differentiates between blue, red and plain or black nodes.

Root nodes as well as immediate sub nodes are blue nodes. This means that they are major items that represent complete entities. In the favorite Web site service, the **`myFavoriteWebSites`** node and the **`favoriteWebSite`** node are considered blue. Each blue node has an **`id`** attribute that can be used to directly query that blue node by its GUID. The sub-nodes however (such as title) are considered red. This means that they represent items that cannot exist on their own. However, you can query those items. For instance, you could retrieve a result set from NMS that contains all the URLs in your favorites without retrieving any other information at all. However, you could not store a URL into the service all by itself. You would have to store a whole **`favoriteWebSite`** entity, which is a blue element.

Black (or plain) elements on the other hand are opaque as far as HSDL is concerned. Those tags are still part of the result set and are validated against the schema, however, you cannot query these tags on their own nor can you query blue or red tags based on the value of black tags. An example is the **`xml:lang`** tag in the favorite Web sites. This tag is retrieved as part of the result set, but you cannot run a query to return the English speaking favorite Web sites only. Therefore, if you wanted to retrieve the URL of the German Microsoft site, you would have to first query all Microsoft sites and then pick the German one locally.

All services support six basic HSDL commands (insert, update, query, replace, delete and subscriptionResponse). However, individual services may add to those commands at will. Let's investigate these basic commands.

Adding Data to a Service

When you start out using NMS, there is no data available. For this reason, it is best to start out with the **`insert`** command. As mentioned before, all HSDL commands are sent to NMS in the form of XML. Here is a simple example to insert a new favorite Web site into the repository:

```
<hs:insertRequest select="/m:myFavoriteWebSites"
   xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
   xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites">
   <m:favoriteWebSite>
      <m:title xml:lang="en">Component Developer Magazine (CoDe)</m:title>
      <m:url>http://www.code-magazine.com/</m:url>
   </m:favoriteWebSite>
</hs:insertRequest>
```

The basic idea is simple: The data to be inserted into the repository is sent as the contents of the **`insertRequest`** tag. To identify where we want to insert that data, we use the **`select`** attribute on that tag (in this case, the destination of the new information is obvious, but with some of the other services, this is not the case). Keep in mind that the top-element you send to NMS has to be a blue element.

Again, the definition of namespaces makes it a bit harder to look at the code. You will have to consult the schema definition to determine which tags are mandatory. As mentioned above, there are a number of tags you may specify with the FavoriteWebSites service, but only the ones shown above are required.

To execute this insert request, you have to post it to NMS. The easiest way to do so is through the **`hspost.exe`** command line utility (we will explore other options later in this white paper). To use **hspost.exe**, save the XML to INSERT.XML and run the following command:

```
hspost.exe –f insert.xml –u username –s myFavoriteWebSites
```

Make sure to replace "username" with an existing user name that has been provisioned on your development server (for more information about provisioning, see the installation section of this white paper).

If everything goes according to plan, you should see a result message similar to the following (wrapped into an additional SOAP envelope, which I omit here):

```
<hs:insertResponse xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
status="success" selectedNodeCount="1" newChangeNumber="208">
   <hs:newBlueId id="E1016A65-BE06-4962-B40B-31393BE6EAC4"/>
</hs:insertResponse>
```

The most important part of this response is the **`status`** attribute, which tells us whether the insert operation was successful.

With this basic information, you should be able to add additional favorites to NMS My FavoriteWebSites.

Querying Data

Now that we added data to our repository, we can query it. Querying data is performed in a similar way as inserting it. First, we need to generate the HSDL command. The following query, stored in QUERY.XML, retrieves all favorite Web sites:

```
<hs:queryRequest
  xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
  xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites">
   <hs:xpQuery select="//m:myFavoriteWebSites" />
</hs:queryRequest>
```

Again, we can execute this query using **hspost.exe**:

```
hspost.exe –f query.xml –u username –s myFavoriteWebSites
```

This query retrieves a result similar to the following:

```
<hs:queryResponse xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites">
   <hs:xpQueryResponse status="success">
      <m:myFavoriteWebSites>
         <m:favoriteWebSite id="C30C45FB-11DA-4CC0-8EE6-70FDD12858D2">
            <m:title xml:lang="en">Microsoft Corp.</m:title>
            <m:url>http://www.microsoft.com/ms.htm</m:url>
         </m:favoriteWebSite>
         <m:favoriteWebSite id="137E8DD0-DB35-4FDC-8C40-238C41A1CF92">
            <m:title xml:lang="en">EPS Software Corp.</m:title>
            <m:url>http://www.eps-software.com/</m:url>
         </m:favoriteWebSite>
      </m:myFavoriteWebSites>
   </hs:xpQueryResponse>
</hs:queryResponse>
```

The reason that we call the query tag **`xpQuery`** is that NMS queries are based on XPath. This means that whatever you send as the value of the select attribute can be a fully qualified XPath query. For instance, if you wanted to retrieve all the URLs in your favorites, you could run the following query:

```
<hs:queryRequest
  xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
  xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites">
   <hs:xpQuery select="//m:myFavoriteWebSites/m:favoriteWebSite/m:url" />
</hs:queryRequest>
```

The result set would be similar to this:

```
<hs:queryResponse xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites">
   <hs:xpQueryResponse status="success">
      <m:url>http://www.microsoft.com/ms.htm</m:url>
      <m:url>http://www.eps-software.com/</m:url>
   </hs:xpQueryResponse>
</hs:queryResponse>
```

You can also filter your result sets according to XPath rules. For instance, if you only wanted to retrieve Microsoft Web sites, you could run the following query:

```
<hs:queryRequest
  xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
  xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites">
   <hs:xpQuery select="//m:favoriteWebSite[m:title='Microsoft Corp.']" />
</hs:queryRequest>
```

Here is the expected result set:

```
<hs:queryResponse xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites">
   <hs:xpQueryResponse status="success">
      <m:favoriteWebSite id="C30C45FB-11DA-4CC0-8EE6-70FDD12858D2">
         <m:title xml:lang="en">Microsoft Corp.</m:title>
         <m:url>http://www.microsoft.com/ms.htm</m:url>
      </m:favoriteWebSite>
   </hs:xpQueryResponse>
</hs:queryResponse>
```

You also have the option of submitting multiple queries at once. Consider this query for instance:

```
<hs:queryRequest
  xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
  xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites">
   <hs:xpQuery select="//m:favoriteWebSite[m:title='Microsoft Corp.']" />
   <hs:xpQuery select="//m:myFavoriteWebSites/m:favoriteWebSite/m:url" />
</hs:queryRequest>
```

This will retrieve multiple result sets embedded within the same query response:

```
<hs:queryResponse xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites">
   <hs:xpQueryResponse status="success">
      <m:favoriteWebSite id="C30C45FB-11DA-4CC0-8EE6-70FDD12858D2">
         <m:title xml:lang="en">Microsoft Corp.</m:title>
         <m:url>http://www.microsoft.com/ms.htm</m:url>
      </m:favoriteWebSite>
   </hs:xpQueryResponse>
   <hs:xpQueryResponse status="success">
      <m:url>http://www.microsoft.com/ms.htm</m:url>
      <m:url>http://www.eps-software.com/</m:url>
   </hs:xpQueryResponse>
</hs:queryResponse>
```

Deleting Information

Deleting information is among the easier tasks. To delete Microsoft's Web site from the list of favorites, post the following command:

```
<hs:deleteRequest
  xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
  xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites"
  select="//m:favoriteWebSite[m:title='Microsoft Corp.']" />
</hs:deleteRequest>
```

Note that in most cases, you probably want to use the **`id`** attribute to select the entity you want to delete. The command above wipes out all of Microsoft's Web sites (well, in our favorites that is).

Replacing Information

To replace existing information, you simply XPath-select a node and provide new information for it:

```
<hs:replaceRequest
   xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core"
   xmlns:m="http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites"
   select="//m:favoriteWebSite[@id='E1016A65-BE06-4962-B40B-31393BE6EAC4']">
   <m:favoriteWebSite>
      <m:title xml:lang="en">CoDe</m:title>
      <m:url>http://www.Code-Magazine.com</m:url>
   </m:favoriteWebSite>
</hs:replaceRequest>
```

This replaces the previously created favorite and replaces its title with the shorter "CoDe."

Other Commands

The commands described so far provide the basic functionality needed to work with NMS. To learn more about other commands (either standard commands or commands specific to a service), please refer to the .NET My Services SDK.

Accessing .NET My Services through SOAP

We have already used SOAP to access NMS when we used **hspost.exe**. However, most of the work was performed for us. All we had to provide was the payload (our HSDL command). **hspost.exe** is a great utility to explore NMS and to test HSDL, but in real world applications, you will have to integrate with NMS more directly. In a pre-VS .NET world, the easiest way to do so is through the Microsoft SOAP Toolkit.

Accessing .NET FavoriteWebSites from VB6

When using SOAP, you have to worry about creating an appropriate SOAP message in addition to HSDL (which is embedded in the SOAP message). The easiest way to do this is to create a SOAP HTTPConnector, then use a serializer to generate the SOAP message, and finally execute it using a SOAP Reader. Here is the code to do so:

```
strSoapNamespace       = "http://schemas.xmlsoap.org/soap/envelope/"
strMessageEncoding     = "UTF-8"

strSoapAction          = "http://schemas.microsoft.com/hs/2001/10/core#request"
strRpNamespace         = "http://schemas.xmlsoap.org/rp/"
strHsCoreNamespace     = "http://schemas.microsoft.com/hs/2001/10/core"
strSecNamespace        = "http://schemas.xmlsoap.org/soap/security/2000-12"

strId                  = "0c69dfd1-6694-11d5-a2bc-00b0d0e9071d"
strDocument            = "content"
```

```
strMethod             = "query"
strGenResponse        = "always"

nUserId               = 1234

'' Create Connector and set properties.
Set Connector = CreateObject("MSSOAP.HttpConnector")
Connector.Property("EndPointURL") = "http://localhost/myFavoriteWebSites"
Connector.Property("SoapAction") =
"http://schemas.microsoft.com/hs/2001/10/core#request"
Connector.BeginMessage

'' Create Serializer.
Set Serializer = CreateObject("MSSOAP.SoapSerializer")
Serializer.Init Connector.InputStream

'' Build .NET My Services SOAP packet.
Serializer.startEnvelope "s", strSoapNamespace, strMessageEncoding
Serializer.startHeader
Serializer.startElement "path", strRpNamespace, , "x"
Serializer.startElement "action", strRpNamespace, , "x"
Serializer.writeString strSoapAction
Serializer.endElement
Serializer.startElement "rev", strRpNamespace, , "x"
Serializer.startElement "via", strRpNamespace, , "x"
Serializer.endElement
Serializer.endElement
Serializer.startElement "to", strRpNamespace, , "x"
Serializer.writeString strTo
Serializer.endElement
Serializer.startElement "id", strRpNamespace, , "x"
Serializer.writeString strId
Serializer.endElement
Serializer.endElement
Serializer.startElement "licenses", strSecNamespace, , "ss"
Serializer.startElement "identity", strHsCoreNamespace, , "h"
Serializer.startElement "kerberos", strHsCoreNamespace, , "h"
Serializer.writeString nUserId
Serializer.endElement
Serializer.endElement
Serializer.endElement
Serializer.startElement "request", strHsCoreNamespace, , "h"
Serializer.SoapAttribute "service", , "myFavoriteWebSites"
Serializer.SoapAttribute "document", ,strDocument
Serializer.SoapAttribute "method", ,strMethod
Serializer.SoapAttribute "genResponse", ,strGenResponse
Serializer.startElement "key", strHsCoreNamespace, , "h"
Serializer.SoapAttribute "instance", ,"0"
Serializer.SoapAttribute "cluster", ,"0"
Serializer.SoapAttribute "puid", ,nUserId
Serializer.endElement
Serializer.endElement
Serializer.endHeader
Serializer.startBody

'' We create the actual body that's the equivalent of the
'' XML file used by the hspost.exe utility...
Serializer.startElement "queryRequest", strHsCoreNamespace, , "hs"
Serializer.SoapAttribute "xmlns:m", ,
"http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites"
Serializer.startElement "xpQuery", strHsCoreNamespace, , "hs"
Serializer.SoapAttribute "select", , "/m:myFavoriteWebSites"
Serializer.endElement
Serializer.endElement

Serializer.endBody
Serializer.endEnvelope
Connector.EndMessage

'' Read in SOAP Response from .NET My Services.
Set Reader = CreateObject("MSSOAP.SoapReader")
Reader.Load Connector.OutputStream

'' Display XML
MsgBox Reader.Body.xml
```

Note the line towards the last section that adds the *queryRequest* tag. This section of the code is where we generate the HSDL message. You can replace this section with any HSDL message you would like. However, make sure you change the *strMethod* variable to "insert" in case you want to send an *insert* command.

Also, make sure to replace the *nUserID* variable with your local, temporary PUID. You can query your PUID using the **hspost.exe** utility in the following fashion:

```
hspost.exe –p
```

This will return a message similar to the following:

```
Your username is markus - PUID = 1234
```

In this case, the PUID/user id is *1234*. This is a temporary PUID assigned to you by ".NET My Services In-A-Box." In a production environment, this is replaced with a user's real PUID (Passport User ID).

Accessing .NET FavoriteWebSites from Visual FoxPro

Accessing NMS from within Visual FoxPro is very similar to accessing it from Visual Basic 6. Some of the syntax differs slightly, but the overall concept and the steps performed are the same. To run the following sample code, create a new form and add a Grid named "grid1." Place the following code in the Init() event:

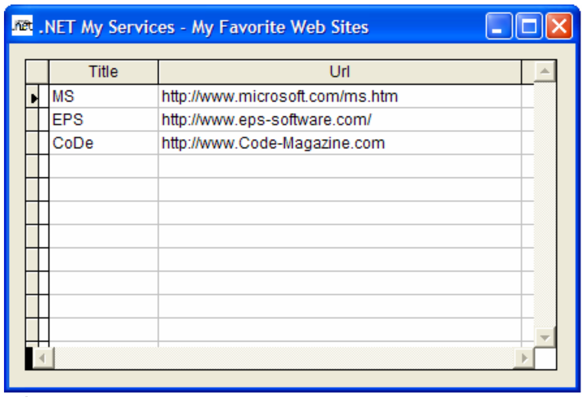
```
LOCAL lcRpNamespace, lcSoapNamespace, lcHsCoreNamespace, lcSecNamespace
LOCAL lcSoapAction, lcUserId
lcRpNamespace          = "http://schemas.xmlsoap.org/rp/"
lcSoapNamespace        = "http://schemas.xmlsoap.org/soap/envelope/"
lcHsCoreNamespace      = "http://schemas.microsoft.com/hs/2001/10/core"
lcSecNamespace         = "http://schemas.xmlsoap.org/soap/security/2000-12"
lcSoapAction    = "http://schemas.microsoft.com/hs/2001/10/core#request"
lcUserId               = "1234"        && Change this to your individual id!!!

** We create a Soap HTTP connector
LOCAL Connection AS MSSOAP.HttpConnector
Connection = CREATEOBJECT("MSSOAP.HttpConnector")
Connection.Property("EndPointURL") = "http://localhost/myFavoriteWebSites"
Connection.Property("SoapAction") =
"http://schemas.microsoft.com/hs/2001/10/core#request"
Connection.BeginMessage

** We use a serializer to create the appropriate SOAP message
LOCAL Serializer AS MSSOAP.SoapSerializer
Serializer = CREATEOBJECT("MSSOAP.SoapSerializer")
```

```
Serializer.Init(Connection.InputStream)

Serializer.startEnvelope( "s", lcSoapNamespace, "UTF-8")
Serializer.startHeader
Serializer.startElement( "path", lcRpNamespace, , "x")
Serializer.startElement( "action", lcRpNamespace, , "x")
Serializer.writeString( lcSoapAction)
Serializer.endElement
Serializer.startElement( "rev", lcRpNamespace, , "x")
Serializer.startElement( "via", lcRpNamespace, , "x")
Serializer.endElement
Serializer.endElement
Serializer.startElement( "to", lcRpNamespace, , "x")
Serializer.writeString( "http://localhost/")
Serializer.endElement
Serializer.startElement( "id", lcRpNamespace, , "x")
Serializer.writeString( "0c69dfd1-6694-11d5-a2bc-00b0d0e9071d")
Serializer.endElement
Serializer.endElement
Serializer.startElement( "licenses", lcSecNamespace, , "ss")
Serializer.startElement( "identity", lcHsCoreNamespace, , "h")
Serializer.startElement( "kerberos", lcHsCoreNamespace, , "h")
Serializer.writeString( lcUserId)
Serializer.endElement
Serializer.endElement
Serializer.endElement
Serializer.startElement( "request", lcHsCoreNamespace, , "h")
Serializer.SoapAttribute( "service", ,"myFavoriteWebSites")
Serializer.SoapAttribute( "document", ,"content")
Serializer.SoapAttribute( "method", ,"query")
Serializer.SoapAttribute( "genResponse", ,"always")
Serializer.startElement( "key", lcHsCoreNamespace, , "h")
Serializer.SoapAttribute( "instance", ,"0")
Serializer.SoapAttribute( "cluster", ,"0")
Serializer.SoapAttribute( "puid", ,lcUserId)
Serializer.endElement
Serializer.endElement
Serializer.endHeader
Serializer.startBody

** We create the actual body that's the equivalent of the
** XML file used by the hspost.exe utility...
Serializer.startElement( "queryRequest", lcHsCoreNamespace, , "hs")
Serializer.SoapAttribute( "xmlns:m", ,
"http://schemas.microsoft.com/hs/2001/10/myFavoriteWebSites")
Serializer.startElement( "xpQuery", lcHsCoreNamespace, , "hs")
Serializer.SoapAttribute( "select", , "/m:myFavoriteWebSites")
Serializer.endElement
Serializer.endElement

Serializer.endBody
Serializer.endEnvelope
Connection.EndMessage

** We are finally ready to post the message to .NET My Services
LOCAL loReader AS MSSOAP.SoapReader
loReader = CREATEOBJECT("MSSOAP.SoapReader")
loReader.Load( Connection.OutputStream )
LOCAL lcXML, loXML
lcXML = loReader.Body.xml

LOCAL loXML as Microsoft.XMLDOM
LOCAL loNode as MICROSOFT.IXMLDOMNode
```

```
loXML = CREATEOBJECT("Microsoft.XMLDOM")
loXML.LoadXML(lcXML)
loNode = loXML.SelectSingleNode("//m:myFavoriteWebSites")

XMLTOCURSOR(loNode.xml,"MyFavoriteWebSites")
THIS.Grid1.RecordSource = "MyFavoriteWebSites"
```

Just as with Visual Basic, the core functionality resides in the six lines of code that insert the **`queryRequest`** node. This is our HSDL query that defines the action to be taken. This part of the query can be replaced with any HSDL needed. Note that the **`method`** attribute on the **`request`** element of the SOAP envelope has to be set to "insert" if you want to run an insert operation.

The key difference between the Visual FoxPro version and the Visual Basic 6 version are the last few lines of code. In the Visual FoxPro version, the returned message body is converted to a VFP cursor using VFP's native XMLToCursor method, and then the resulting table is bound to the grid. **Figure 3** shows the result.



Title	Url
MS	http://www.microsoft.com/ms.htm
EPS	http://www.eps-software.com/
CoDe	http://www.Code-Magazine.com

Figure 3

Note: Ensure that you replace the **`lcUserID`** variable at the very beginning with the PUID that's assigned to you temporarily. You can retrieve the temporary PUID using **hspost.exe** in the following manner:

```
hspost.exe –p
```

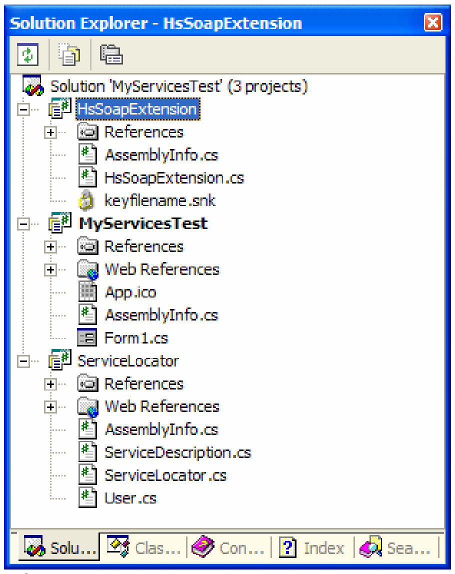
Accessing .NET My Services with Visual Studio .NET

When using Visual Studio .NET, accessing NMS works somewhat different. The basic principles are still the same. However, most of the tasks are abstracted away. A set of wrapper objects sit between the developer and the XMI, making it much easier and more straightforward to access NMS.

Note: At this point, there is very little documentation available about the NMS wrapper classes for Visual Studio .NET. Many of the classes will change significantly before NMS releases!

Microsoft provides assemblies that assist the developer in accessing NMS, such as the *HSSoapExtensions* assembly and the *ServiceLocator* assembly. The ServiceLocator assembly provides the proxy classes specific to NMS. The HSSoapExtensions assembly deals with NMS-specific SOAP requirements, such as encrypting and decrypting the SOAP requests.

As a first simple example, create a new C# Windows application. Immediately proceed to import the HSSoapExtensions and ServiceLocator projects, as shown in **Figure 4**.

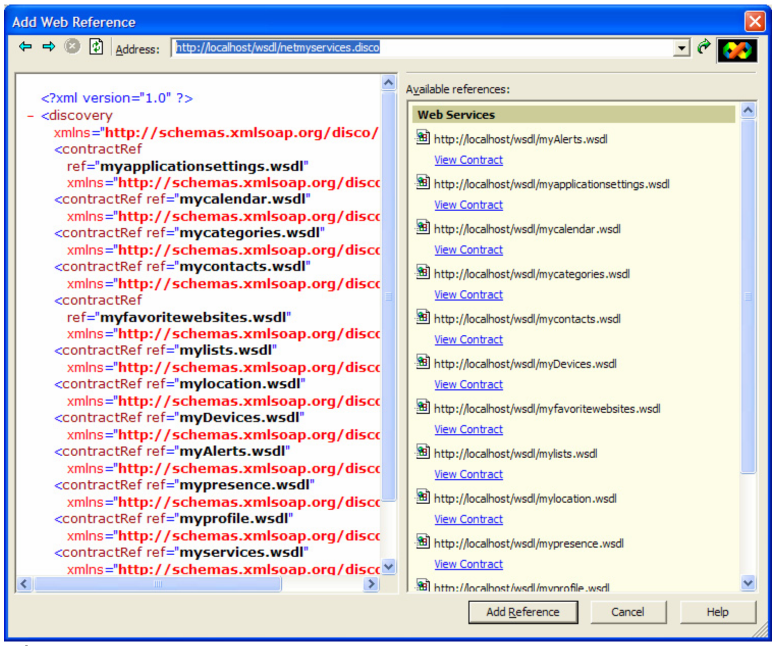


```
Solution 'MyServicesTest' (3 projects)
  HsSoapExtension
    References
    AssemblyInfo.cs
    HsSoapExtension.cs
    keyfilename.snk
  MyServicesTest
    References
    Web References
    App.ico
    AssemblyInfo.cs
    Form1.cs
  ServiceLocator
    References
    Web References
    AssemblyInfo.cs
    ServiceDescription.cs
    ServiceLocator.cs
    User.cs
```

Figure 4

Also, add references to both assemblies to your project's references.

Then, proceed to add a Web Reference to the NMS services installed on your local machine. To do so, right-click on your test project (the main one you created, not those imported) and select **Add Web Reference…**. Now browse to http://localhost/wsdl/NETMyServices.disco. You will see the dialog displayed in **Figure 5**.



http://localhost/wsdl/netmyservices.disco

```
  <?xml version="1.0" ?>
- <discovery
    xmlns="http://schemas.xmlsoap.org/disco/
    <contractRef
      ref="myapplicationsettings.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef ref="mycalendar.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef ref="mycategories.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef ref="mycontacts.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef
      ref="myfavoritewebsites.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef ref="mylists.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef ref="mylocation.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef ref="myDevices.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef ref="myAlerts.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef ref="mypresence.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef ref="myprofile.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
    <contractRef ref="myservices.wsdl"
      xmlns="http://schemas.xmlsoap.org/discc
```

Available references:

Web Services

http://localhost/wsdl/myAlerts.wsdl
View Contract
http://localhost/wsdl/myapplicationsettings.wsdl
View Contract
http://localhost/wsdl/mycalendar.wsdl
View Contract
http://localhost/wsdl/mycategories.wsdl
View Contract
http://localhost/wsdl/mycontacts.wsdl
View Contract
http://localhost/wsdl/myDevices.wsdl
View Contract
http://localhost/wsdl/myfavoritewebsites.wsdl
View Contract
http://localhost/wsdl/mylists.wsdl
View Contract
http://localhost/wsdl/mylocation.wsdl
View Contract
http://localhost/wsdl/mypresence.wsdl
View Contract
http://localhost/wsdl/myprofile.wsdl

Figure 5

View the contract of the *myFavoriteWebSites* service and add a reference by clicking **Add Reference**. This adds a new Web reference, called *localhost*, to your project. Normally, we would rename that reference, but for this example, we will retain that name.

Now, open your main Web form for editing. Add a Treeview control, which we will use to display the retrieved favorites. Then, open the code view and add the following references to the top of the source:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using Microsoft.Hs.ServiceLocator;
using Microsoft.Hs;
using MyServicesTest.localhost;
```

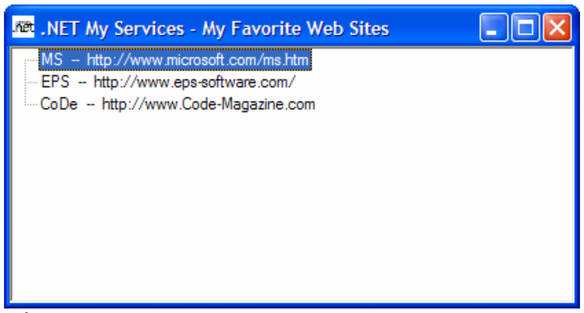
Note: Replace `MyServicesTest` with the name of your project.

Then add the following code to the form's Load() event:

```
private void Form1_Load(object sender, System.EventArgs e)
{
   // We get a reference to the favorite web sites service...
   ServiceLocator oHS = new ServiceLocator("Http://localhost/myServices",
      "c:\\HsSoapExtension.log",true);
   myFavoriteWebSites myFav =
      (myFavoriteWebSites)oHS.GetService(typeof(myFavoriteWebSites),
      User.GetCurrentUser());

   // We query all the favorite web sites...
   queryRequestType queryRequest = new queryRequestType();
   xpQueryType xpQuery = new xpQueryType();
   xpQuery.select="/m:myFavoriteWebSites/m:favoriteWebSite";
   queryRequest.xpQuery = new xpQueryType[]{xpQuery};
   queryResponseType queryResponse = myFav.query(queryRequest);

   // We add the items to the tree...
   favoriteWebSiteType returnedWebSite;
   for (int i = 0; i<queryResponse.xpQueryResponse[0].Items.Length; i++)
   {
      returnedWebSite =
         (favoriteWebSiteType) queryResponse.xpQueryResponse[0].Items[i];

         TreeNode current = new TreeNode(returnedWebSite.title[0].Value+
            "   --   "+returnedWebSite.url);
         this.treeView1.Nodes.Add(current);
   }
}
```

First, the *ServiceLocator* object instantiates a reference to NMS and subsequently to the *MyFavoriteWebSites* service. The *myFav* object is the provided wrapper object specific to the *MyFavoriteWebSites* service.

Then, we create a new *queryRequest* object as well as an *xpQueryType* object. We set the select property of the *xpQuery* object to retrieve favorite Web sites based on an XPath query. Note that this construct maps 1:1 to the HSDL XML structure we would pass through SOAP (which is what happens behind the scenes). We attach the *xpQuery* object to the *queryRequest* object (this is a collection and we could attach as many queries as we need), and fire the query through the *myFav* proxy object. This generates a *queryResponse* object. This object has a collection of query responses, and each of those has a collection of favorite Web sites. We can now simply access properties on those objects and add corresponding items to the Treeview on our form, as shown in **Figure 6**.



MS -- http://www.microsoft.com/ms.htm
EPS -- http://www.eps-software.com/
CoDe -- http://www.Code-Magazine.com

Figure 6

It is much easier to access NMS with Visual Studio .NET than it is with plain SOAP. You don't have to worry about any of the underlying SOAP details. However, the object structure is closely tied to HSDL structures. For this reason, knowledge of the HSDL is very beneficial, even in Visual Studio .NET.