DYNAMIC HOST CONFIGURATION PROTOCOL WEB SERVICES

by

Jason Rupard

A project submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

April 2008

The project Dynamic Host Configuration Protocol Web Services submitted by Jason Rupard in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by:	Date
Yap S. Chua Project Director	
Charles N. Winton Graduate Director of the School of Computing	
Judith L. Solano Director of the School of Computing	

ACKNOWLEDGEMENT

I want to thank Dr. Yap S. Chua for all his extra time and commitment in helping me over the last several years. I would also like to thank my family; especially my wife Regeana, whose support and encouragement have helped me complete my Masters work.

CONTENTS

TABLES

FIGURES

ABSTRACT

Dynamic Host Configuration Protocol (DHCP) Web Services offer significant improvements to the management of Microsoft's DHCP server. The web services utilize Simple Object Access Protocol (SOAP), providing an open and standardized interface for DHCP server management, and implement the Web Service Interoperability (WS-I) Basic Profile 1.1, a set of non-proprietary web service specifications. This standardized communication protocol allows network administrators and developers to manage Microsoft's DHCP server, in ways not achievable with previous management tools provided by Microsoft, or other outside sources, chiefly because the malleable nature of the open framework allows users to form their own management tools, as they see fit. The DHCP Web Services include two features innovative to Microsoft's DHCP Server: an authorization model to control access of DHCP server objects, and a search to query the DHCP server for client lease information using regular expressions. Open management, authorization, and search create a flexible and adaptable tool set for managing a DHCP server, thus making it more appealing to large network environments with complex needs.

Chapter 1

IMPETUS

In a professional IT environment, a lack of management flexibility within Microsoft's Dynamic Host Configuration Protocol (DHCP) server product is noticeable. The DHCP server is a crucial part of any IP based network. Therefore, its management is crucial to the operation of that network. One could say, the management of a DHCP server is the management of client devices on an IP based network.

The lack of adaptability of Microsoft's DHCP server stems from a deficiency in its management options or tools, produced by Microsoft or other outside sources. This inadequacy makes it unappealing for IT professionals to choose Microsoft's DHCP server for their large and complex network environments. In these environments, real world problems require administrators have several alternatives to manage such an important piece of networking infrastructure. The ability to produce immediate functionality is sometimes needed to help manage these complex networks, providing administrators the ability to take a problem and solve it quickly with additional methods not native to the DHCP server. In this regard, Microsoft's DHCP server has definite room for improvement when it comes to large networks.

The DHCP server needs an open management framework to provide administrators the freedom to develop new functionalities, as well as design new integration points for other network or non-network applications, while also granting accessibility to information housed in the DHCP server itself. An open management framework could offer these features, along with the ability to produce creative solutions for future problems, making it especially flexible. An authorization model, which would ultimately add to the adaptability of the DHCP server by allowing delegation of server access and management to multiple users, not solely administrators, would also prove to be potentially beneficial.

1.1 Background

DHCP allows IP networked devices (clients) to obtain unique IP addresses and other network parameters from a DHCP server [Droms97, Droms03]. Among the most common network parameters are default gateway, network mask, and DNS server addresses. DHCP created a centralized paradigm for IP address and network parameter management, allowing administrators to configure and reconfigure these network parameters in one location, where DHCP clients communicate to the server and receive new or updated network settings.

A DHCP server manages IP addresses and configurations for one or more subnets, which means clients on those subnets will receive their IP addresses and configurations from that DHCP server. IP addresses are allocated dynamically to clients through a lease, which has an associated length of time before expiration. The client device must check back with the DHCP server to renew its lease, before the lease is up, at which time, the client could receive updated network configurations and a different IP address. An IP reservation is a special type of lease that allows a client, based on its hardware address, to receive the same IP address each time it renews its lease. This allows DHCP servers to assign static IP addresses to clients. However, clients with static IP addresses must still check back with the DHCP server, because the options of the lease may require changes. DHCP options are network parameters passed from the server to the client devices [Alexander97]. Option 6 is an example of a DHCP option, which refers to an array of DNS server addresses. This option will let the client know what the network's DNS addresses are, allowing domain name resolution to occur. The remainder of this paper will refer to the DHCP subnets, leases, reservations, and options as DHCP objects.

1.2 Research

Given the deficiencies of the DHCP server, there is potential for improvement, which led to this research. First, it needed to be ascertained if anyone else had already produced a remedy for the DHCP server's shortcomings. Initial research confirmed the lack of management options. The current tools provided by Microsoft include a management console GUI and CLI [Technet05]. While both fully manage the DHCP server, they are not very open, which means they do not have standardized interfaces with which potential consumers could manage the server. In addition, the provided GUI and CLI do not provide any search or authorization capabilities. Management tools from the open source community were non-existent. However, there are a couple of vendor tools, the most viable of which belongs to Mice and Men, who provide the standard functionality of a DHCP management tool, along with an authorization model that allows non-administrative users to manage different objects within a DHCP server [Mice&Men07]. The Mice and Men product is still not as flexible as desired, since it lacks the openness a standardized communication method could provide.

For open management of the DHCP server, SOAP HTTP web services [W3C00], was examined and chosen to provide a standardized and open interface. Use of the .NET framework [MSDN08A], specifically ASP.NET [MSDN08B], was also decided upon, since it is the preferred Microsoft development platform. Another reason for choosing this technology was research showed ASP.NET could implement the Web Service Interoperability standard (WS-I) Basic Profile 1.1 [MSDN08C]. WS-I defines web services in such a way that different development platforms can interface in a standardized manner, a requirement for any open framework [WS-I06].

Another aspect of this research focused on how to interface directly with the DHCP server itself. Microsoft provides a C library, DHCP Server API (DHCPSAPI) [MSDN08D], to manage their DHCP server. The DHCP server can be totally configured through this API; meaning configurations are stored in a database and can be manipulated in real time. A means to bridge the language gap between Microsoft's C library and the .NET framework was found in Platform Invoke (p/invoke), which allows .NET classes to consume unmanaged code DLLs, in this case DHCPSAPI [MSDN08E].

Though most of this research focused on facilitating an open management framework, an API needed to be found that would help in the implementation of the authorization paradigm that needed to be created from scratch. Microsoft's Authorization Manager (AzMan) [McPherson04, McPherson06] fit nicely, because it is accessible through the .NET framework and uses an XML file to store authorization data. AzMan can also take advantage of existing user authentication stores, such as Active Directory, by mapping those users to role-based authorizations created by a developer [Reimer03]. Another appealing reason for using AzMan was that it provides a scope construct, which makes possible hierarchically based access control over objects within the DHCP server. For all of these reasons, AzMan together was chosen as the model to help implement authorization with DHCP Web Services.

Chapter 2

CONTEXT

Three innovations to Microsoft's DHCP Server management suite were developed: open management via SOAP, a search capability that would grant information access, and an authorization layer for access control. While all three concepts deserve special attention, the open management paradigm is the largest and most significant of the three, the latter two are important functions within it. These three new capabilities should significantly increase the flexibility and appeal of Microsoft's DHCP server.

2.1 Open Management

The open management interface is provided through SOAP. SOAP is a standardized messaging protocol usually transported via HTTP. This open interface can be considered a Service Oriented Architecture (SOA), in which the DHCP server management functions are offered in the form of services, allowing users to consume those services for a particular need. SOAP is a protocol implemented by most development platforms, including .NET, Java, and Perl. It consists of XML-based messages that are passed to and from web services and their clients. While SOAP is a standardized protocol, small implementation variations within a development platform can lead to web service and consumer communication issues. WS-I is needed to constrain SOAP with implementation and interoperability guidelines, so platforms will be able to communicate seamlessly.

The open management framework developed as part of this project is provided through ASP.NET web services that use SOAP and conform to the WS-I Basic Profile 1.1. This open service architecture is the most important feature of the DHCP Web Services project, because SOAP combined with WS-I to create a standardized method for management tools to interface with Microsoft's DHCP server. This allows administrators the ability to be creative when solving DHCP related issues. An administrator can develop solutions based in any platform that implements SOAP and conforms to WS-I. This interface allows network and business processes the ability to interact, share data, and manage DHCP objects through a shared integration point. Web service consumers could consist of administrative scripts, management console tools, or integrated network and business applications. One example of an integrated network application would have the DHCP Web Service in a subordinate role of a larger IP Address Management (IPAM) system. It could create an IP reservation, while the IPAM makes other changes in DNS, and possibly routers and/or switches, to complete its IPAM operations. This example shows just one of the countless added functions the open management interface could facilitate.

2.2 Search

In addition to the open management interface, a search feature within the DHCP Web Services was created, to gather information easily from the DHCP server. The search function queries client leases belonging to a DHCP server or subnet, and matches those clients based on three possible attributes: hostname, IP address, or MAC address. To match an attribute within a client lease, the search function uses .NET regular expressions [MSDN08F]. The use of regular expressions allows complex search patterns to match client leases.

Searching is important because it allows information gathering and auditing processes to occur, which alternately makes the DHCP server more functional. Security could use this function to execute audit tasks or to gather forensics information. Additionally, a network application could use this functionality to create a network switch database, in which MAC addresses are mapped to known IP addresses in the DHCP server. In this application, MAC addresses would be collected from network switches, and their corresponding IP addresses from the DHCP server, to create relational database structures. Network administrators could then query these relational databases, to provide client IP address and network switch relationship information.

2.3 Authorization

The third innovation and second function added to the management system, the authorization layer of DHCP Web Services, allows scope-sensitive, role-based access controls over the DHCP server. In general, this means it allows user accounts various access rights to DHCP objects within the server that normally only a server administrator could manipulate. More specifically, authorization scopes were defined, in this application, at three levels in a DHCP server: at the server, subnet, and IP range levels. Each of these scopes contains three possible role assignments: Owner, Manager, and Auditor. User accounts are assigned roles within scopes, which means those accounts would have particular rights to DHCP objects, for a particular layer of the DHCP server, but not for others. Therefore, if a user is assigned the Manager role, he is allowed to read and update DHCP objects, but not create nor delete those objects. Moreover, because this authorization layer is scope sensitive, if that same user is authorized to manage only one subnet, he can only read and update objects on that specific subnet. It is important to mention these authorization scopes are hierarchical in nature, which means the user can manage all DHCP objects underneath the highest layer for which he has authorization. Using the previous example, the user authorized to manage only a subnet also has access to the lower IP range level. In addition to the previously mentioned authorization scopes is one specialized scope that contains two special Global roles: Global Administrator and Global Auditor. These roles have full control or read only access, respectively, over all servers managed through DHCP Web Services. Users in the Global Administrator role are the only ones who can manage authorizations, making them the users who actually delegate access rights to DHCP objects.

Authorization is important because it allows network administrators the ability to delegate DHCP objects to organizational subunits or individual users, saving time and money on tedious tasks subunits may want to manage themselves. Giving subunits power allows network administrators to focus on other important operations. Allowing a user the rights to manage a set of IP addresses for printers is an example application. The user could then set reservations based on the printer's MAC address without contacting the network administrator, saving time and eliminating a possible bottleneck in the process. Allowing a user to manage an entire subnet of a DHCP server is another example. Before this ability, network administrators might consider installing an additional DHCP server, just to let that user manage that subnet. Now, the administrator can save resources and money, by just allowing the user to manage only that subnet within the same DHCP server. This idea can be extrapolated to multiple subnets with different users managing each. Adding an authorization layer to DHCP Web Services, increased the functionality of the DHCP server, thus making it far more appealing to potential clients.

The authorization layer, the search function, and the open architecture show the flexibility that has been achieved with DHCP Web Services. The combination of these three functions allows network administrators to carry out more complex tasks and gives them a multitude of options on how to do so, thus giving the Microsoft DHCP server the adaptability needed to thrive in large complex network environments.

Chapter 3

SOFTWARE DEVELOPMENT

To create DHCP Web Services, five hierarchical layers of software coding were designed, as well as two supplemental sample clients, to display its functionality. The five layers of DHCP Web Services use C# and the .NET Framework. Each software layer builds upon the last, with each of the five levels interfacing with the neighboring layers above and below. Beginning with a foundation of unmanaged code interoperability, called the Interop layer, each successive layer was built, ending with the final layer, SOAP Web Services.

The Interop is thus titled, because it uses the p/invoke technique, to exchange and utilize data with the DHPC server API, DHCPSAPI, which is a C API library. DHCP Server Management Objects (SMO), is the second layer of the Web Services. It provides a set of friendly .NET classes that manage a DHCP server and interface directly with the Interop's C-like structures and functions. The third layer, named DHCP Server Operations, uses the SMO layer to create a set of static DHCP operations. The forth layer, Authorized DHCP Server Operations, maps directly to functions in the DHCP Server Operations layer, with the addition of access controls for the unique authorization paradigm of DHCP Web Services. Again, these operations are static, which will complement the final layer, SOAP Web Services. This layer presents the Authorized Server Operations of the previous layer, as SOAP web services using ASP.NET. Figure 1 provides a diagram of the software layers of DHCP Web Services.

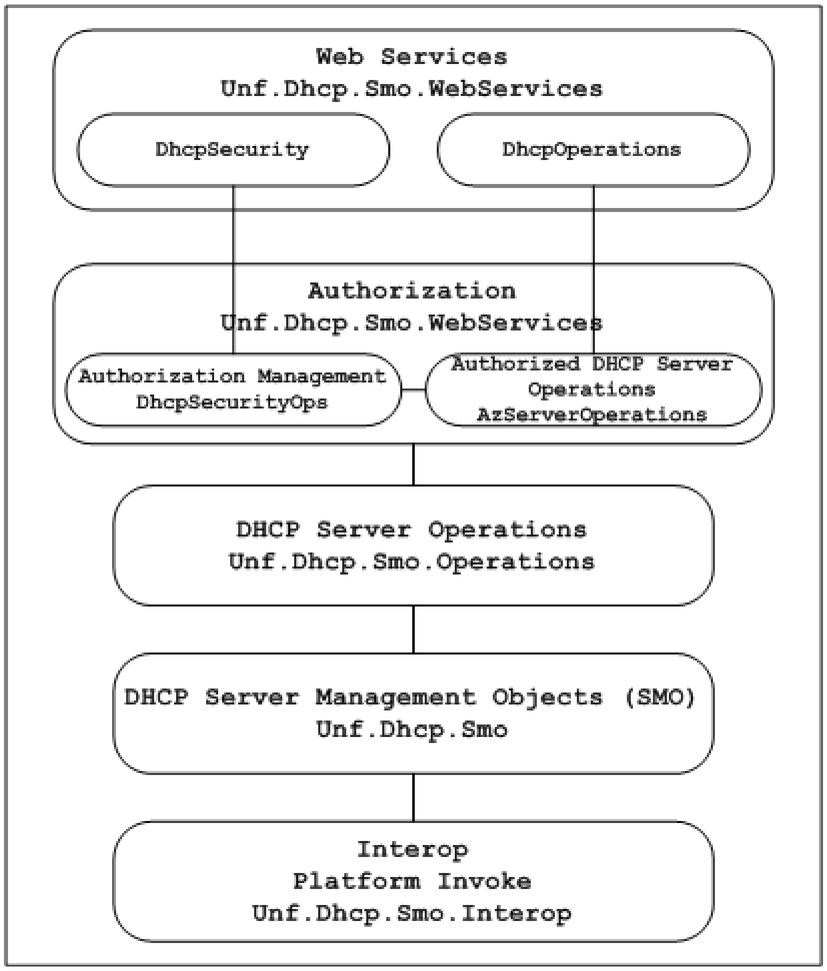

Figure 1: DHCP Web Services Software Layers

In addition to the software layers, two web service clients were developed, meant to demonstrate the DHCP Web Services. The first client, written in C# .NET, fully consumes the functionality of DHCP Web Services. The second sample client was written in PERL on a Linux platform, to exhibit the interoperability of DHCP Web Services.

3.1 Interop

The Interop software layer, known in the code base as the namespace `Unf.Dhcp.Smo.Interop`, is the foundation of DHCP Web Services. The DHCP server API defines functions, structures, and enumerations used to manage a DHCP server. Because the DHCP server API is in C and this project was written in C# .NET, this layer was created, as an interoperability interface between the two. This layer can also be referred to as a p/invoke wrapper. This wrapper defines C# prototypes, structures, and enumerations, to mimic its C library counterpart, using a .NET utility namespace called `System.Runtime.InteropServices`. Classes within this namespace allow .NET managed code to make external calls to C functions, which are exported in Dynamic Link Libraries (DLLs). The namespace contains a `Marshal` class that facilitates data type conversions to and from both C platform dependent types and .NET Framework types. This process is known as marshalling. The `Marshal` class also contains functions that allocate unmanaged memory, a normal operation when developing in C. When using p/invoke, developers have to manage memory, as if they were developing in C, since .NET does not provide a garbage collection system for this type of service. This is why a memory management class was created, called `MemManager`. `MemManager` manages memory allocations for p/invoke calls, along with the proper release of that memory, after use. In addition, the .NET `Marshal` class automatically does some marshaling of data types, so developers do not have to worry about memory allocations or type conversions in certain situations. For example, a C `char*` is automatically converted to a .NET `String` type, by the `Marshal` class. The `Marshal` class does not automatically handle structures, unions, and most pointer types. Instead, it provides helper functions to facilitate the marshaling of these types. Unions are one of the more difficult structures with which to deal, because C# does not have a construct for a union. However, the `Marshal` class does define special attributes to help mimic a C union, the most important of which is the `FieldOffset` attribute. This attribute allows a developer to define the byte location where a structure member starts, relative to the explicit size of that structure in memory. With this mechanism, a developer can make all union members start at the same byte offset, thus mimicking a C union in C#. This technique was used on more than one occasion in the Interop layer, since the DHCP server API defines a couple of structures containing unions.

By using the previously described techniques of p/invoke, a C# wrapper class was created. It contains all known and documented functions exported by the DHCP server API DLL, thus allowing an exchange of data between C and C#. C# `extern` prototypes of the DHCPSAPI functions were created in the `NativeMethods` class within the `Unf.Dhcp.Smo.Interop` namespace. For example, the `NativeMethods.DhcpSetOptionValueV5` method directly references its counterpart in the external DHCPSAPI DLL. When a call to this method occurs, the system marshals the C# parameters to C data types, pushes them onto the call stack, and then executes the DHCPSAPI DLL C function. C# structures, located in the

NativeStructs.cs source file, were created to mimic the C function parameters used by DHCPSAPI. `NativeMethods`, `MemManager`, and native structures make up the `Unf.Dhcp.Smo.Interop` namespace. These are used by the next layer, DHCP Server Management Objects, which abstracts low-level details of the C like types and functions.

3.2 DHCP Server Management Objects (SMO)

Server Management Objects were created to interface directly with the Interop layer, forming the .NET DHCP management classes of DHCP Web Services. This layer hides implementation details of the archaic p/invoke code of the Interop layer, presenting a well-organized .NET interface to the above software layers. Consumers of this layer need not worry about memory management or type specific details of the DHCP server API. Instead, developers can concentrate on using the SMO for flexible DHCP management, at the .NET managed code level. This layer contains an almost one-to-one mapping of DHCPSAPI native types and function calls, to SMO classes and methods. For example, the native type `DHCP_CLIENT_INFO` is mapped to the `DhcpClient` class in the `Unf.Dhcp.Smo` namespace, and the `DhcpClient.Set` method maps to the native function, `DhcpSetClientInfo`. This technique is extended to all native types and functions in the DHCP server API library. The SMO layer is a set of .NET classes used to manage a DHCP server. The next layer will use these classes to form a set of static DHCP operations suitable for web service consumption.

3.3 DHCP Server Operations

The DHCP Server Operations layer, or the `Unf.Dhcp.Smo.Operations` namespace in the code base, is a set of static DHCP operations that facilitate full management of the DHCP server. These operations usually combine two or more SMO method calls to complete their task, ultimately providing an abstraction to the details of the SMO layer. For example, the `CreateSubnet` operation makes three SMO calls; one to create the subnet's information in the DHCP database, one to create the default address pool of that subnet, and one to set the default Dynamic DNS option value for that subnet.

The DHCP Server Operations layer is also where the search functions of DHCP Web Services were implemented. These search functions are implemented at two DHCP levels, server and subnet, with two types of data returns, making four search functions all together. At the subnet level, the search function enumerates all client leases belonging to that subnet, matching leases based on a regular expression pattern. The search function does the same at the server level, but instead of searching only one subnet, it searches all of the server's subnets. The search function can return all leases it finds, or it can return just the first lease it finds, forgoing any other matching. The four search functions are known in the code base as `DhcpServerFindOne`, `DhcpServerFindAll`, `DhcpSubnetFindOne`, and `DhcpSubnetFindAll`. Each function uses .NET regular expressions to match client leases, based on hostname, IP address, or MAC address. In addition, these patterns can be combined to create a more detailed search filter. Search functions execute a logical AND condition on non-null search patterns. For example, for a search filter with a hostname pattern equal to “`a.*`”, a MAC address pattern equal to “`.*A3`”, and the IP address pattern null or empty, the search function would return all client leases where the client hostname begins with the letter “a” and the MAC address ends with byte “A3”. The search functions round out the DHCP Server Operations layer, giving way to the next layer, which implements the newly created authorization functionality.

3.4 Authorization

The authorization layer and the final web services layer of DHCP Web Services are tightly coupled, with both existing in the same namespace: `Unf.Dhcp.Smo.WebServices`. The idea behind this was to allow developers unfettered access to the DHCP Server Operations layer, if they did not need the authorization or web services layers. While this may seem counterintuitive to the purpose of the project, this is another option an administrator or developer might appreciate, when it comes to DHCP Web Services as a whole. This option leaves open the opportunity for a developer to create his own management tool using the SMO library.

Based in the authorization layer, the Authorized DHCP Server Operations, or `AzServerOperations` class in the `Unf.Dhcp.Smo.WebServices` namespace, are one-to-one function mappings of the DHCP Server Operations layer, with added access controls. This layer uses two methods to enforce these access controls. The first is a direct allow/deny method. If a user does not have sufficient authorization access to complete an operation, like `CreateSubnet`, the operation will throw an unauthorized exception. Otherwise, the operation will allow the creation to occur. The second method is security trimming. In this approach, DHCP operations that return a set of DHCP objects are "trimmed," to exclude objects to which the user does not have access. For example, if a user has read access to a range of IP addresses within a subnet, and executes the `EnumClients` operation, the operation would normally return all client leases within that subnet. However, with security trimming, the caller will only receive client leases within the range of IP addresses to which he has been granted read access. Furthermore, the set of client leases could be empty, if the user does not have access to read any leases within a subnet.

3.4.1 AzMan

The authorization layer of DHCP Web Services uses the Authorization Manager framework, AzMan, to help implement the unique authorization model developed for the DHCP server. When the DHCP Web Services application starts up, it loads an authorization data store in the form of an XML file. This store contains data describing authorization scopes, roles, and operations, which are created, updated, and deleted by DHCP Web Services, using the AzMan .NET classes. Scopes are implemented through AzMan, with unique string identifiers that give them context. Server scope identifier strings were created in the form "`SRV <SIP>`" where `SIP` is the DHCP server's IP address. Subnet scopes are identified as "`SRV <SIP> - SN`

<NIP>” where NIP is the network IP address of the subnet. IP range scopes are identified as “SRV <SIP> - SN <NIP> - IPR <BIP>:<EIP>” where BIP is the beginning IP address in the range and EIP is the end IP address in the range. Each scope was designed to consist of three roles (Owner, Manager, and Auditor), and will be created on demand, when the first user account is assigned to one of the three roles. In AzMan, these roles are assigned application operations that represent privileged actions. Four operations were created, generally representing the operations that can occur, when managing a DHCP server: Create, Read, Update, and Delete. The Owner role is assigned all operations. The Manager role is assigned Read and Update. The Auditor role is assigned only the Read operation. A scheme was created, to assign privileged actions to DHCP operations, based on obvious associations. For example, a user must be assigned a role having the ability to execute the Create operation, when that user calls the CreateReservation DHCP operations. There is also a special internal role called Limited, assigned a special internal Traverse operation. This role is automatically assigned at the parent subnet level to a user of an IP range scope assignment. This is analogous to the traverse permission on a file system. The user needs to be able to traverse the subnet in order to read the IP range within that subnet. Table 1 shows the list of DHCP operations, with the minimum access role and scope assignments needed to execute each operation.

DHCP Operation	**Role**	**Scope**
EnumSubnets	Limited	Subnet
CreateSubnet	Owner	Subnet
DeleteSubnet	Owner	Subnet
UpdateSubnet	Manager	Subnet
UpdateSubnetRange	Manager	Subnet
CreateSubnetExclusion	Manager	Subnet
DeleteSubnetExclusion	Manager	Subnet
SubnetFindAllClients	Auditor	IP Range
SubnetFindOneClient	Auditor	IP Range
ServerFindAllClients	Auditor	Server
ServerFindOneClient	Auditor	Server
EnumPools	Auditor	Subnet
EnumLeases	Auditor	IP Range
CreateLease	Owner	IP Range
UpdateLease	Manager	IP Range
DeleteLease	Owner	IP Range
CreateReservation	Owner	IP Range
UpdateReservation	Manager	IP Range
DeleteReservation	Owner	IP Range
EnumReservations	Auditor	IP Range
SetOptionValue	Manager	Scope Specific
RemoveOptionValue	Manager	Scope Specific
EnumOptionValues	Auditor	Scope Specific
GetOptionValue	Auditor	Scope Specific

Table 1: DHCP Operations with Minimum Access Roles and Scopes

Finally, the `GlobalAdministrator` role is assigned all operations and the `Global Auditor` role is assigned the read operation to all DHCP servers managed through DHCP Web Services. To allow local administrators of the Windows Server hosting DHCP Web Services to manage fully DHCP servers and their authorizations, bootstrap and failsafe mechanisms were placed in the web services source code. This allows first runtime access to DHCP Web Services, so an administrator can start to delegate access controls to other user accounts, as needed.

The authorization structures just described were implemented in the `DhcpSecurityOps` class, within the `Unf.Dhcp.Smo.WebServices` namespace. In addition, this class contains static methods to add and remove authorizations. It also contains other utilities, such as, the `isAdmin` method that returns a Boolean value based on whether the current user is in the `GlobalAdministrator` role. DHCP Web Services uses the `DhcpSecurityOps` class to manage DHCP authorizations, thus allowing access to the authorization framework through the DHCP Web Services' open framework.

3.5 Web Services

The open management framework becomes operational in the web services layer. As the fifth and final software layer, it implements SOAP web services via ASP.NET. This layer contains two sets of web services that are both defined in the `Unf.Dhcp.Smo.WebServices` namespace. The first set of services, known as `DhcpSecurity` web services, maps to the static methods created in the `DhcpSecurityOps` class of the authorization layer. The second set of web services, named `DhcpOperations`, is used to manage a DHCP server. This set of services maps directly to static functions defined in the `AzServerOperations` class, within the authorization layer. Both sets of web services, `DhcpSecurity` and `DhcpOperations`, implement SOAP messages, based on the WS-I standard. In addition, each set provides a browser-based set of HTML pages that developers can visit to learn more about the services. The documentation included in these pages contains example SOAP messages that can be passed to and from the web services.

This documentation is a built-in function of ASP.NET web services, and can be very useful to developers. The ASP.NET web services also generate a Web Services Description Language file, WSDL, which defines standardized SOAP XML objects, operations, and service endpoints. Most development platforms can read WSDL files and generate corresponding "stub" classes, used to consume the web services. A properly formatted WSDL file is a requirement of the WS-I standard. The source code for DHCP Web Services is included in Appendix A.

3.6 Runtime Environment

The DHCP Web Services runtime environment is ASP.NET 2.0, on a Windows Server 2003 IIS 6.0 web server. The IIS web server performs authentication of user accounts to the web services. To authenticate users, IIS was configured to use the basic authentication standard of HTTP, in conjunction with SSL, to encrypt communication of usernames and passwords over the network. The IIS web server checks a username and password against the server's local authentication store, or Active Directory domain, then finally passes the validated user account to DHCP Web Services. DHCP Web Services then checks authorization access controls, to validate the user has permission to execute an operation. Next, if the user is permitted, DHCP Web Services uses a trusted subsystem approach to execute operations on the DHCP server [MSDN08G], which means it uses a privileged service account to execute the operation on the users' behalf. This service account has full administrative privileges to the DHCP servers managed by DHCP Web Services. If the user does not have proper access to execute an operation, the service account will do nothing and the web services will send back a SOAP fault. The service account is configured during the installation steps of the DHCP Web Services application. These steps are outlined as part of the installation documentation included in Appendix B

3.7 Clients

3.7.1 .NET Client

Known as the DWS Management Console, this .NET client fully utilizes the capability of the DHCP Web Services. Implemented using C# .NET and Microsoft's Management Console (MMC) 3.0 API, the GUI based management tool allows administrators to manage multiple DHCP servers in their network environments. Installation and user manual documentation for the DHCP Web Services MMC can be found in Appendix C.

3.7.2 Perl Samples

A set of Perl 5.8 code samples were created. These can be executed from a Linux platform. These samples demonstrate the interoperability achieved in this project. The sample Perl scripts use the module `SOAP::Lite`, which is a Perl API for consuming SOAP web services. These scripts exhibit the functionality of DHCP server operations, `CreateSubnet`, `SetOptionValue`, `CreateReservation`, `DeleteSubnet`, `DeleteReservation`, and `ServerFindAllClients`. These functions cover the combinations of input and output types used in DHCP Web Services. The Perl samples have been included in Appendix D.

Chapter 4

CONCLUSION

As the DHCP server is crucial to an IP based network, a deficient management system for it could severely stifle the productivity of that network. Such a deficiency in Microsoft's DHCP server product was found and eliminated by creating DHCP Web Services, an open management framework including both authorization and search functionality. By using SOAP and conforming to the WS-I interoperability standards when creating the DHCP Web Services, a standardized integration point was provided, allowing open communication between the DHCP server and potential users. Administrators and developers can now interface with this integration point in many creative and dynamic ways to fit their specific needs, as opposed to being limited to the minimal management options previously offered.

To create DHCP Web Services, five hierarchical software layers were designed: Interop, SMO, DHCP Server Operations, Authorization, and Web Services. The main innovations of this project were the open framework, authorization model, and search function. However, the Interop and SMO layers are also breakthroughs, since neither Microsoft, another vendor, nor the open source community has provided a .NET DHCP server management library. Extra steps were taken to document the DHCP SMO library, so developers could use it in the future, without the use of DHCP Web Services. However, the added authorization and search functions developed are far more instrumental in improving the flexibility of Microsoft’s DHCP server. The authorization paradigm allows delegation of DHCP access rights, potentially providing other technical individuals the opportunity to perform tasks in which they have a vested interest. This ability could save clients time, money, and resources, which obviously makes authorization a desirable addition to the minimal offerings of Microsoft’s management options. These options also lack any search operations that clients could use for information gathering, security, or troubleshooting situations. The search function included in DHCP Web Services uses regular expression filters to help fill this gap. Because it provides a powerful tool to query the DHCP server’s client database, this search function provides the flexibility needed by any comprehensive DHCP server management tool set.

These three innovations have transformed Microsoft’s DHCP server from one more suited to a small network, to one viable for a large enterprise network. Before this project, Microsoft and other available tools for DHCP management consisted of ordinary proprietary tools not open in any way. Therefore, they could not be molded to fit a user’s needs. The open nature of DHCP Web Services, along with the search and authorization functions, provide an almost infinite amount of possible solutions for its owners. This form of Service Orientated Architecture changes management of a DHCP server into a non-vendor-specific DHCP management service on the network. Consumers from different computer platforms can now manage the DHCP service, despite the fact it is provided by Microsoft. The SOA approach also adds longevity to the DHCP management service, which could become a legacy application without openness and standardizations. Given the lack of tools and options existing before this project, DHCP Web Services is a significant accomplishment. DHCP Web Services provides flexibility and adaptability not realized until this point. This means administrators can now seriously consider the use of Microsoft’s DHCP server, in their large and complex networks.

REFERENCES

Print Publications:

[Droms02]
Droms, R., and T. Lemon, The DHCP Handbook, 2d ed., Sams, Indianapolis, 2002.

[Reimer03]
Reimer, S., and M. Mulcare, Active Directory for Microsoft Windows Server 2003 Technical Reference, Microsoft Press, Redmond, 2003.

Electronic Sources:

[Droms97]
Droms, R., “Dynamic Host Configuration Protocol,” RFC 2132, http://tools.ietf.org/html/rfc2131, 1997, last accessed March 29, 2008.

[Alexander97]
Alexander, S. and R. Droms, “DHCP Options and BOOTP Vendor Extensions,” RFC 2132, http://tools.ietf.org/html/rfc2132, 1997, last accessed March 29, 2008.

[McPherson06]
McPherson, D., and D. Crawford, “Developing Applications Using Windows Authorization Manager,” http://msdn2.microsoft.com/en-us/library/aa480244.aspx, 2006, last accessed March 29, 2008.

[McPherson04]
McPherson, D., “Role-Based Access Control for Multi-tier Applications Using Authorization Manager,” http://technet2.microsoft.com/WindowsServer/en/library/72b55950-86cc-4c7f-8fbf-3063276cd0b61033.mspx, 2004, last accessed March 29, 2008.

[Mice&Men07]
Mice and Men DHCP Management Module, http://www.menandmice.com/solutions/modules/dhcpmgmt, 2007, last access March 29, 2008.

[MSDN08A]
MSDN: .Net Framework, http://msdn2.microsoft.com/en-us/netframework/, 2008, last accessed March 29, 2008.

[MSDN08B]
MSDN: ASP.NET, http://msdn2.microsoft.com/en-us/netframework/aa336522.aspx, 2008, last accessed March 29, 2008.

[MSDN08C]
MSDN: Building WS-I Basic Profile Compliant Web Services Using ASP.NET 2.0, http://msdn2.microsoft.com/en-us/library/ms230196(VS.80).aspx, 2008, last accessed March 29, 2008.

[MSDN08D]
MSDN: DHCP Server Management API, http://msdn2.microsoft.com/en-us/library/aa363376(VS.85).aspx, last revision February 2008, last accessed March 29, 2008.

[MSDN08E]
MSDN: Consuming Unmanaged DLL Functions, http://msdn2.microsoft.com/en-us/library/26thfadc(VS.80).aspx, 2008, last accessed March 29, 2008.

[MSDN08F]
MSDN: .NET Framework Regular Expressions, http://msdn2.microsoft.com/en-us/library/hs600312.aspx, 2008, last accessed March 29, 2008.

[MSDN08G]
MSDN: Trusted Subsystem, http://msdn2.microsoft.com/en-us/library/aa480587.aspx, 2005, last accessed March 29, 2008.

[Technet05]
TechNet: DHCP tools, http://technet2.microsoft.com/windowsserver/en/library/c7801a5c-721e-4dff-a917-28610296360e1033.mspx, last revision January 2005, last accessed March 29, 2008.

[W3C00]
Box, D., et al., "Simple Object Access Protocol (SOAP) 1.1," http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, 2000, last accessed March 29, 2008.

[WS-I06]
Ballinger, K., et al., "Basic Profile Version 1.1," http://www.ws-i.org/Profiles/BasicProfile-1.1.html, last revision April 2006, last accessed March 29, 2008.

APPENDIX A

DHCP Web Services Code Listing

```
/*
 * DHCP Server Management Objects
 *
 *  File: AzServerOperations.cs
 *  Namespace: Unf.Dhcp.Smo.WebServices
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using Microsoft.Interop.Security.AzRoles;

using Unf.Dhcp.Smo;
using Unf.Dhcp.Smo.Operations;

namespace Unf.Dhcp.Smo.WebServices
{
    public class AzServerOperations
    {

        #region Subnet Ops...
        //requires limited on subnet
        public static DhcpSubnet[] EnumSubnets(DhcpIpAddress server)
        {
            List<DhcpSubnet> allSubnets = ServerOperations.EnumSubnets(server);
            List<DhcpSubnet> trimedSubnets = new List<DhcpSubnet>();

            IAzClientContext2 ctx = DhcpSecurityOps.GetClientContext();
            DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
            azSubnet.Server = server;

            foreach (DhcpSubnet subnet in allSubnets)
            {
                azSubnet.Subnet = subnet.Address;
                if (azSubnet.AccessCheck("EnumSubnets", ctx, DhcpAzOps.Traverse))
                    trimedSubnets.Add(subnet);
            }

            return trimedSubnets.ToArray();
        }

        //requires create on server+
        public static void CreateSubnet(DhcpIpAddress server, DhcpSubnet network,
DhcpIpRange ipRange)
        {
            DhcpAzServerScope azServer = new DhcpAzServerScope();
            azServer.Server = server;

            if (azServer.AccessCheck("CreateSubnet",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Create))
                ServerOperations.CreateSubnet(server, network, ipRange);
            else
                throw new UnauthorizedAccessException("CreateSubnet Access Denied");
        }

        //requires delete on server+
        public static void DeleteSubnet(DhcpIpAddress server, DhcpIpAddress network)
        {
            DhcpAzServerScope azServer = new DhcpAzServerScope();
            azServer.Server = server;

            if (azServer.AccessCheck("DeleteSubnet",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Delete))
            {
                ServerOperations.DeleteSubnet(server, network);

                DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
                azSubnet.Server = server;
                azSubnet.Subnet = network;
                DhcpSecurityOps.RemoveAzSubnetAzIpRanges(azSubnet);
            }
            else
                throw new UnauthorizedAccessException("DeleteSubnet Access Denied");
        }

        //requires update on subnet+
        public static void UpdateSubnet(DhcpIpAddress server, DhcpSubnet network)
        {
            DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
            azSubnet.Server = server;
            azSubnet.Subnet = network.Address;

            if (azSubnet.AccessCheck("UpdateSubnet",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Update))
                ServerOperations.UpdateSubnet(server, network);
            else
                throw new UnauthorizedAccessException("UpdateSubnet Access Denied");

        }

        //requires update on subnet+
        public static void UpdateSubnetRange(DhcpIpAddress server, DhcpIpAddress
network, DhcpIpRange ipRange)
        {
            DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
            azSubnet.Server = server;
            azSubnet.Subnet = network;

            if (azSubnet.AccessCheck("UpdateSubnetRange",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Update))
                ServerOperations.UpdateSubnetRange(server, network, ipRange);
            else
                throw new UnauthorizedAccessException("UpdateSubnetRange Access
Denied");
        }

        //requires update on subnet+
        public static void CreateSubnetExclusion(DhcpIpAddress server, DhcpIpAddress
network, DhcpIpRange ipExRange)
        {
            DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
            azSubnet.Server = server;
            azSubnet.Subnet = network;

            if (azSubnet.AccessCheck("CreateSubnetExclusion",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Update))
                ServerOperations.CreateSubnetExclusion(server, network, ipExRange);
            else
                throw new UnauthorizedAccessException("CreateSubnetExclusion Access
Denied");

        }

        //requires update on subnet+
        public static void DeleteSubnetExclusion(DhcpIpAddress server, DhcpIpAddress
network, DhcpIpRange ipExRange)
        {
            DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
            azSubnet.Server = server;
            azSubnet.Subnet = network;

            if (azSubnet.AccessCheck("DeleteSubnetExclusion",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Update))
                ServerOperations.DeleteSubnetExclusion(server, network, ipExRange);
            else
                throw new UnauthorizedAccessException("DeleteSubnetExclusion Access
Denied");

        }

        //requires read on ipr+
        public static DhcpClient[] SubnetFindAllClients(DhcpIpAddress server,
DhcpIpAddress network, DhcpClientFilter terms)
        {
            List<DhcpClient> allClients =
ServerOperations.SubnetFindAllClients(server, network, terms);
            List<DhcpClient> trimedClients = new List<DhcpClient>();

            IAzClientContext2 ctx = DhcpSecurityOps.GetClientContext();
            DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
            azIpRange.Server = server;
            azIpRange.Subnet = network;

            foreach (DhcpClient client in allClients)
            {
                azIpRange.StartIp = client.IpAddress;
                if (azIpRange.AccessCheck("SubnetFindAllClients", ctx,
DhcpAzOps.Read))
                    trimedClients.Add(client);
            }

            return trimedClients.ToArray();
        }

        //requires read on ipr+
        public static DhcpClient SubnetFindOneClient(DhcpIpAddress server,
DhcpIpAddress network, DhcpClientFilter terms)
        {
            DhcpClient oneClient = ServerOperations.SubnetFindOneClient(server,
network, terms);
            DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
            azIpRange.Server = server;
            azIpRange.Subnet = network;
            azIpRange.StartIp = oneClient.IpAddress;

            if (azIpRange.AccessCheck("SubnetFindOneClient",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Read))
                return oneClient;
            else
                return null;
        }

        //requires read on server+
        public static DhcpClient[] ServerFindAllClients(DhcpIpAddress server,
DhcpClientFilter terms)
        {
            List<DhcpClient> allClients =
ServerOperations.ServerFindAllClients(server, terms);

            IAzClientContext2 ctx = DhcpSecurityOps.GetClientContext();
            DhcpAzServerScope azServer = new DhcpAzServerScope();
            azServer.Server = server;
            if (azServer.AccessCheck("ServerFindAllClients", ctx, DhcpAzOps.Read))
                return allClients.ToArray();
            else
                throw new UnauthorizedAccessException("ServerFindAllClients Access 
Denied, Requires Read on Server");

        }

        //requires read on server+
        public static DhcpClient ServerFindOneClient(DhcpIpAddress server, 
DhcpClientFilter terms)
        {
            DhcpClient oneClient = ServerOperations.ServerFindOneClient(server, 
terms);
            DhcpAzServerScope azServer = new DhcpAzServerScope();
            azServer.Server = server;

            if (azServer.AccessCheck("ServerFindOneClient", 
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Read))
                return oneClient;
            else
                throw new UnauthorizedAccessException("ServerFindAllClients Access 
Denied, Requires Read on Server");
        }

        //requires read on subnet+
        public static DhcpIpRange[] EnumPools(DhcpIpAddress server, DhcpIpAddress 
network)
        {
            DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
            azSubnet.Server = server;
            azSubnet.Subnet = network;

            if (azSubnet.AccessCheck("EnumPools", DhcpSecurityOps.GetClientContext(), 
DhcpAzOps.Read))
                return ServerOperations.EnumPools(server, network);
            else
                return null;
        }
        #endregion

        #region Lease Ops...

        //requries ip+ read
        public static DhcpClient[] EnumLeases(DhcpIpAddress server, DhcpIpAddress 
network)
        {
            List<DhcpClient> allClients = ServerOperations.EnumLeases(server, 
network);
            List<DhcpClient> trimedClients = new List<DhcpClient>();

            IAzClientContext2 ctx = DhcpSecurityOps.GetClientContext();
            DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
            azIpRange.Server = server;
            azIpRange.Subnet = network;

            foreach (DhcpClient client in allClients)
            {
                azIpRange.StartIp = client.IpAddress;
                if (azIpRange.AccessCheck("EnumLeases", ctx, DhcpAzOps.Read))
                    trimedClients.Add(client);
            }

            return trimedClients.ToArray();
        }

        //requires ip+ create
        public static void CreateLease(DhcpIpAddress server, DhcpClient lease)
        {
            DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
            azIpRange.Server = server;
            azIpRange.Subnet = new DhcpIpAddress(lease.IpAddress.GetUIntAddress() &
lease.SubnetMask.GetUIntAddress());
            azIpRange.StartIp = lease.IpAddress;

            if (azIpRange.AccessCheck("CreateLease",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Create))
                ServerOperations.CreateLease(server, lease);
            else
                throw new UnauthorizedAccessException("CreateLease Access Denied");
        }

        //requires ip+ update
        public static void UpdateLease(DhcpIpAddress server, DhcpClient lease)
        {
            DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
            azIpRange.Server = server;
            azIpRange.Subnet = new DhcpIpAddress(lease.IpAddress.GetUIntAddress() &
lease.SubnetMask.GetUIntAddress()); ;
            azIpRange.StartIp = lease.IpAddress;

            if (azIpRange.AccessCheck("UpdateLease",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Update))
                ServerOperations.UpdateLease(server, lease);
            else
                throw new UnauthorizedAccessException("UpdateLease Access Denied");
        }

        //requires ip+ delete
        public static void DeleteLease(DhcpIpAddress server, DhcpIpAddress network,
DhcpIpAddress leaseIp)
        {
            DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
            azIpRange.Server = server;
            azIpRange.Subnet = network;
            azIpRange.StartIp = leaseIp;

            if (azIpRange.AccessCheck("DeleteLease",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Delete))
                ServerOperations.DeleteLease(server, leaseIp);
            else
                throw new UnauthorizedAccessException("DeleteLease Access Denied");
        }

        #endregion

        #region Rez Ops...
        //requires ip+ create
        public static void CreateReservation(DhcpIpAddress server, DhcpIpAddress
network, DhcpReservation res)
        {
            DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
            azIpRange.Server = server;
            azIpRange.Subnet = network;
            azIpRange.StartIp = res.ReservedIp;

            if (azIpRange.AccessCheck("CreateReservation",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Create))
                ServerOperations.CreateReservation(server, network, res);
            else
                throw new UnauthorizedAccessException("CreateReservation Access
Denied");
        }

        //requires ip+ update
        public static void UpdateReservation(DhcpIpAddress server, DhcpIpAddress
network, DhcpReservation res)
        {
            DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
            azIpRange.Server = server;
            azIpRange.Subnet = network;
            azIpRange.StartIp = res.ReservedIp;

            if (azIpRange.AccessCheck("UpdateReservation",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Update))
                ServerOperations.UpdateReservation(server, network, res);
            else
                throw new UnauthorizedAccessException("UpdateReservation Access
Denied");
        }

        //requires ip+ delete
        public static void DeleteReservation(DhcpIpAddress server, DhcpIpAddress
network, DhcpIpAddress resIp)
        {
            DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
            azIpRange.Server = server;
            azIpRange.Subnet = network;
            azIpRange.StartIp = resIp;

            if (azIpRange.AccessCheck("DeleteReservation",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Delete))
                ServerOperations.DeleteReservation(server, network, resIp);
            else
                throw new UnauthorizedAccessException("DeleteReservation Access
Denied");
        }

        //requires ip+ read
        public static DhcpReservation[] EnumReservations(DhcpIpAddress server,
DhcpIpAddress network)
        {
            List<DhcpReservation> allRezz = ServerOperations.EnumReservations(server,
network);
            List<DhcpReservation> trimedRezz = new List<DhcpReservation>();

            IAzClientContext2 ctx = DhcpSecurityOps.GetClientContext();
            DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
            azIpRange.Server = server;
            azIpRange.Subnet = network;

            foreach (DhcpReservation rez in allRezz)
            {
                azIpRange.StartIp = rez.ReservedIp;
                if (azIpRange.AccessCheck("EnumReservations", ctx, DhcpAzOps.Read))
                    trimedRezz.Add(rez);
            }

            return trimedRezz.ToArray();
        }
        #endregion

        #region Option Ops...

        //requires update on contexted based scope
        public static void SetOptionValue(DhcpIpAddress server, DhcpOptionValue value)
        {
            DhcpAzScope contextScope = null;
            switch (value.OptionScope.type)
            {
                case DhcpOptionScopeType.Server:
                    DhcpAzServerScope azServer = new DhcpAzServerScope();
                    azServer.Server = server;
                    contextScope = azServer;
                    break;
                case DhcpOptionScopeType.Subnet:
                    DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
                    azSubnet.Server = server;
                    azSubnet.Subnet = value.OptionScope.SubnetIp;
                    contextScope = azSubnet;
                    break;
                case DhcpOptionScopeType.Reservation:
                    DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
                    azIpRange.Server = server;
                    azIpRange.Subnet = value.OptionScope.SubnetIp;
                    azIpRange.StartIp = value.OptionScope.ReservationIp;
                    contextScope = azIpRange;
                    break;
            }

            if (contextScope.AccessCheck("SetOptionValue",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Update))
                ServerOperations.SetOptionValue(server, value);
            else
                throw new UnauthorizedAccessException("SetOptionValue Access Denied");
        }

        //requires update on contexted based scope
        public static void RemoveOptionValue(DhcpIpAddress server, DhcpOptionValue
value)
        {
            DhcpAzScope contextScope = null;
            switch (value.OptionScope.type)
            {
                case DhcpOptionScopeType.Server:
                    DhcpAzServerScope azServer = new DhcpAzServerScope();
                    azServer.Server = server;
                    contextScope = azServer;
                    break;
                case DhcpOptionScopeType.Subnet:
                    DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
                    azSubnet.Server = server;
                    azSubnet.Subnet = value.OptionScope.SubnetIp;
                    contextScope = azSubnet;
                    break;
                case DhcpOptionScopeType.Reservation:
                    DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
                    azIpRange.Server = server;
                    azIpRange.Subnet = value.OptionScope.SubnetIp;
                    azIpRange.StartIp = value.OptionScope.ReservationIp;
                    contextScope = azIpRange;
                    break;
            }

            if (contextScope.AccessCheck("RemoveOptionValue",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Update))
                ServerOperations.RemoveOptionValue(server, value);
            else
                throw new UnauthorizedAccessException("RemoveOptionValue Access
Denied");
        }

        //requires read on contexted based scope
        public static DhcpOptionValue[] EnumOptionValues(DhcpIpAddress server,
DhcpOptionClassType classType, DhcpOptionScope scope)
        {
            DhcpAzScope contextScope = null;
            switch (scope.type)
            {
                case DhcpOptionScopeType.Server:
                    DhcpAzServerScope azServer = new DhcpAzServerScope();
                    azServer.Server = server;
                    contextScope = azServer;
                    break;
                case DhcpOptionScopeType.Subnet:
                    DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
                    azSubnet.Server = server;
                    azSubnet.Subnet = scope.SubnetIp;
                    contextScope = azSubnet;
                    break;
                case DhcpOptionScopeType.Reservation:
                    DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
                    azIpRange.Server = server;
                    azIpRange.Subnet = scope.SubnetIp;
                    azIpRange.StartIp = scope.ReservationIp;
                    contextScope = azIpRange;
                    break;
            }

            if (contextScope.AccessCheck("EnumOptionValues",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Read))
                return ServerOperations.EnumOptionValues(server, classType, scope);
            else
                return null;
        }

        //requires read on contexted based scope
        public static DhcpOptionValue GetOptionValue(DhcpIpAddress server, UInt32
optionId, DhcpOptionClassType classType, DhcpOptionScope scope)
        {
            DhcpAzScope contextScope = null;
            switch (scope.type)
            {
                case DhcpOptionScopeType.Server:
                    DhcpAzServerScope azServer = new DhcpAzServerScope();
                    azServer.Server = server;
                    contextScope = azServer;
                    break;
                case DhcpOptionScopeType.Subnet:
                    DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
                    azSubnet.Server = server;
                    azSubnet.Subnet = scope.SubnetIp;
                    contextScope = azSubnet;
                    break;
                case DhcpOptionScopeType.Reservation:
                    DhcpAzIpRangeScope azIpRange = new DhcpAzIpRangeScope();
                    azIpRange.Server = server;
                    azIpRange.Subnet = scope.SubnetIp;
                    azIpRange.StartIp = scope.ReservationIp;
                    contextScope = azIpRange;
                    break;
            }

            if (contextScope.AccessCheck("GetOptionValue",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Read))
                return ServerOperations.GetOptionValue(server, optionId, classType,
scope);
            else
                return null;
        }

        //only requires access to DWS itself
        public static DhcpOption[] EnumOptions(DhcpIpAddress server,
DhcpOptionClassType classType)
        {
            //DhcpAzServerScope azServer = new DhcpAzServerScope();
            //azServer.Server = server;

            //if (azServer.AccessCheck("EnumOption",
DhcpSecurityOps.GetClientContext(), DhcpAzOps.Read))
                return ServerOperations.EnumOptions(server, classType);
            //else
             //   return null;
        }
        #endregion

    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpAzAccount.cs
 *  Namespace: Unf.Dhcp.Smo.WebServices
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;

namespace Unf.Dhcp.Smo.WebServices
{
    public class DhcpAzAccount
    {
        public String AccountName;
        public DhcpAzRole Role;

        public DhcpAzAccount(){}

        public DhcpAzAccount(String accountName, DhcpAzRole role)
        {
            this.AccountName = accountName;
            this.Role = role;
        }
    }

    public class DhcpAzGlobalAccount
    {
        public String AccountName;
        public DhcpAzGlobalRole GlobalRole;

        public DhcpAzGlobalAccount(){}

        public DhcpAzGlobalAccount(String accountName, DhcpAzGlobalRole role)
        {
            this.AccountName = accountName;
            this.GlobalRole = role;
        }
    }

    public enum DhcpAzGlobalRole
    {
        GlobalAuditor = 0,
        GlobalAdministrator = 1
    }

    public enum DhcpAzRole
    {
        Auditor = 1,
        Manager = 2,
        Owner = 3
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpAzScope.cs
 *  Namespace: Unf.Dhcp.Smo.WebServices
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Text.RegularExpressions;
using Microsoft.Interop.Security.AzRoles;

namespace Unf.Dhcp.Smo.WebServices
{
    public abstract class DhcpAzScope
    {
        public DhcpAzScopeType ScopeType;
        public DhcpAzScope() { }

        public abstract String GenerateAzScopeName() ;

        public abstract Boolean AccessCheck(String opDesc, IAzClientContext2 azClient,
DhcpAzOps operation);
    }

    public class DhcpAzServerScope : DhcpAzScope
    {
        public DhcpIpAddress Server;
        public DhcpAzServerScope() { this.ScopeType = DhcpAzScopeType.Server; }

        public override string GenerateAzScopeName()
        {
            if (this.Server == null)
                throw new InvalidOperationException();
            return "SRV " + this.Server.ToString();
        }

        protected static string GenerateAzScopeName(DhcpIpAddress server)
        {
            return "SRV " + server.ToString();
        }

        public override Boolean AccessCheck(String opDesc, IAzClientContext2 azClient,
DhcpAzOps operation)
        {
            Object[] result;
            Object[] azOp = new Object[] { operation };
            String[] scopes = new String[] { "", this.GenerateAzScopeName() };

            foreach (String scope in scopes)
            {
                try
                {
                    result = (Object[])azClient.AccessCheck(opDesc, new Object[] {
scope }, azOp, null, null, null, null, null);
                    if ((int)result[0] == 0)
                        return true;
                }
                catch { }
            }

            return false;
        }
    }

    public class DhcpAzSubnetScope : DhcpAzServerScope
    {
        public DhcpIpAddress Subnet;
        public DhcpAzSubnetScope() { this.ScopeType = DhcpAzScopeType.Subnet; }

        public override string GenerateAzScopeName()
        {
            if (this.Server == null || this.Subnet == null)
                throw new InvalidOperationException();
            return "SRV " + this.Server.ToString() + " - SN " +
this.Subnet.ToString();
        }
        protected static string GenerateAzScopeName(DhcpIpAddress server,
DhcpIpAddress subnet)
        {
            return "SRV " +server.ToString() + " - SN " + subnet.ToString();
        }

        public override Boolean AccessCheck(String opDesc, IAzClientContext2 azClient,
DhcpAzOps operation)
        {
            Object[] result;
            Object[] azOp = new Object[] { operation };
            String[] scopes = new String[] { "",
DhcpAzServerScope.GenerateAzScopeName(this.Server), this.GenerateAzScopeName() };

            foreach(String scope in scopes)
            {
                try
                {
                    result = (Object[])azClient.AccessCheck(opDesc, new Object[] {
scope }, azOp, null, null, null, null);
                    if ((int)result[0] == 0)
                        return true;
                }
                catch { }
            }

            return false;
        }
    }

    public class DhcpAzIpRangeScope : DhcpAzSubnetScope
    {
        public DhcpIpAddress StartIp;
        public DhcpIpAddress EndIp;
        public DhcpAzIpRangeScope() { this.ScopeType = DhcpAzScopeType.IpRange; }

        public override string GenerateAzScopeName()
        {
            if (this.Server == null || this.Subnet == null || this.StartIp == null ||
this.EndIp == null)
                throw new InvalidOperationException();
            return "SRV " + this.Server.ToString() + " - SN " + this.Subnet.ToString()
+
                " - IPR " + this.StartIp + ":" + this.EndIp;
        }

        public override Boolean AccessCheck(String opDesc, IAzClientContext2 azClient,
DhcpAzOps operation)
        {
            Object[] result;
            Object[] azOp = new Object[] { operation };
            String[] scopes = new String[] { "",
DhcpAzServerScope.GenerateAzScopeName(this.Server),
                DhcpAzSubnetScope.GenerateAzScopeName(this.Server, this.Subnet) };//,
this.GenerateAzScopeName() };

            //check scope hierarchy
            foreach (String scope in scopes)
            {
                try
                {
                    result = (Object[])azClient.AccessCheck(opDesc, new Object[] {
scope }, azOp, null, null, null, null, null);
                    if ((int)result[0] == 0)
                        return true;
                }
                catch { }
            }
            //find all scope this user belongs to
            Object cursor = null;
            Object[] clientScopes = (Object[])azClient.GetAssignedScopesPage(0, 2000,
ref cursor);
            Regex filter = new Regex(@"^" + scopes[2] + @"\s-\sIPR\s(\d.*):(\d.*)$");
            foreach (String clientScope in clientScopes)
            {
                Match m = filter.Match(clientScope);
                if (m.Success)
                {
                    DhcpIpAddress startIp = new DhcpIpAddress(m.Groups[1].Value);
                    DhcpIpAddress endIp = new DhcpIpAddress(m.Groups[2].Value.Trim());

                    //isbetween?  if so, check this scope to see if operation allowed
                    if (startIp.GetUIntAddress() <= this.StartIp.GetUIntAddress() &&
                     this.StartIp.GetUIntAddress() <= endIp.GetUIntAddress())
                    {
                        try
                        {
                            result = (Object[])azClient.AccessCheck(opDesc, new
Object[] { clientScope }, azOp, null, null, null, null, null);
                            if ((int)result[0] == 0)
                                return true;
                        }
                        catch { }
                    }
                }
            }

            return false;
        }
    }

    public enum DhcpAzScopeType
    {
        IpRange = 0,
        Subnet = 1,
        Server = 2
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpClient.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;

using Unf.Dhcp.Smo.Interop;

namespace Unf.Dhcp.Smo
{
    public class DhcpClient
    {
        public DhcpIpAddress IpAddress;
        public DhcpIpAddress SubnetMask;
        public DhcpMacAddress MacAddress;
        public String Name;
        public String Comment;
        public DateTime LeaseExpires;

        public DhcpClient() { }
        public void Create(DhcpIpAddress server)
        {
            using (MemManager mem = new MemManager())
            {
                DHCP_CLIENT_INFO info = this.ConvertToNative(mem);

                uint Response = NativeMethods.DhcpCreateClientInfo(server.ToString(),
ref info);

                if (Response != 0)
                    throw new DhcpException(Response);
            }

        }

        public void Update(DhcpIpAddress server)
        {
            using (MemManager mem = new MemManager())
            {
                DHCP_CLIENT_INFO info = this.ConvertToNative(mem);

                uint Response = NativeMethods.DhcpSetClientInfo(server.ToString(), ref
info);

                if (Response != 0)
                    throw new DhcpException(Response);
            }
        }

        public static void Delete(DhcpIpAddress server, DhcpSearchInfo term)
        {
            using (MemManager mem = new MemManager())
            {
                DHCP_SEARCH_INFO sinfo = term.ConvertToNative(mem);

                uint Response = NativeMethods.DhcpDeleteClientInfo(server.ToString(),
ref sinfo);

                if (Response != 0)
                    throw new DhcpException(Response);
            }
        }

        public static DhcpClient Get(DhcpIpAddress server, DhcpSearchInfo term)
        {
            IntPtr iPtr;
            DhcpClient client = new DhcpClient();
            using (MemManager mem = new MemManager())
            {
                DHCP_SEARCH_INFO sinfo = term.ConvertToNative(mem);

                uint Response = NativeMethods.DhcpGetClientInfo(server.ToString(), ref
sinfo, out iPtr);

                if (Response != 0)
                {
                    if (iPtr != IntPtr.Zero)
                        NativeMethods.DhcpRpcFreeMemory(iPtr);
                    throw new DhcpException(Response);
                }

                DHCP_CLIENT_INFO cinfo =
(DHCP_CLIENT_INFO)Marshal.PtrToStructure(iPtr, typeof(DHCP_CLIENT_INFO));
                client.IpAddress = new DhcpIpAddress(cinfo.ClientIpAddress);
                client.SubnetMask = new DhcpIpAddress(cinfo.SubnetMask);
                client.Name = cinfo.ClientName;
                client.Comment = cinfo.ClientComment;
                client.LeaseExpires = cinfo.ClientLeaseExpires.Convert();
                byte[] mac = new byte[cinfo.ClientHardwareAddress.DataLength];
                Marshal.Copy(cinfo.ClientHardwareAddress.Data, mac, 0, mac.Length);
                client.MacAddress = new DhcpMacAddress(mac);

                if (iPtr != IntPtr.Zero)
                    NativeMethods.DhcpRpcFreeMemory(iPtr);
            }
            return client;
        }

        public static List<DhcpClient> EnumAll(DhcpIpAddress server, DhcpIpAddress
subnet)
        {
            UInt32 Response = 0, ResumeHandle = 0;
            UInt32 nRead = 0, nTotal = 0;
            IntPtr retPtr, iPtr;
            List<DhcpClient> clients = new List<DhcpClient>();
            DHCP_CLIENT_INFO nativeClient;
            DHCP_CLIENT_INFO_ARRAY nativeArray;

            for ( ; ; )
            {
                Response = NativeMethods.DhcpEnumSubnetClients(server.ToString(),
subnet.GetUIntAddress(),
                    ref ResumeHandle, 65536, out retPtr, out nRead, out nTotal);

                //ERROR_MORE_DATA = 234
                if (Response != 0 && Response != 234 || retPtr == IntPtr.Zero)
                {
                    if (retPtr != IntPtr.Zero)
                        NativeMethods.DhcpRpcFreeMemory(retPtr);
                    throw new DhcpException(Response);
                }

                //work
                nativeArray = (DHCP_CLIENT_INFO_ARRAY)Marshal.PtrToStructure(retPtr,
typeof(DHCP_CLIENT_INFO_ARRAY));
                for (int i = 0; i < nativeArray.NumElements; ++i)
                {
                    iPtr = Marshal.ReadIntPtr(nativeArray.Elements, (i *
Marshal.SizeOf(typeof(IntPtr))));
                    nativeClient = (DHCP_CLIENT_INFO)Marshal.PtrToStructure(iPtr,
typeof(DHCP_CLIENT_INFO));
                    clients.Add(DhcpClient.ConvertFromNative(nativeClient));
                }

                //free on last successful call
                if (Response == 0)
                {
                    NativeMethods.DhcpRpcFreeMemory(retPtr);
                    break;
                }
            }
            return clients;
        }

        internal DHCP_CLIENT_INFO ConvertToNative(MemManager Mem)
        {
            if (this.IpAddress == null || this.SubnetMask == null || this.MacAddress
== null
                || this.LeaseExpires == null)
                throw new InvalidOperationException("Client invalid");

            DHCP_CLIENT_INFO info = new DHCP_CLIENT_INFO();
            info.ClientIpAddress = this.IpAddress.GetUIntAddress();
            info.SubnetMask = this.SubnetMask.GetUIntAddress();
            info.ClientName = this.Name;
            info.ClientComment = this.Comment;
            Byte[] mac = this.MacAddress.GetByteArray();
            DHCP_CLIENT_UID uid = new DHCP_CLIENT_UID();
            uid.DataLength = (uint)mac.Length;
            uid.Data = Mem.AllocByteArray(this.MacAddress.GetByteArray());
            info.ClientHardwareAddress = uid;

            info.ClientLeaseExpires = new DATE_TIME(this.LeaseExpires);

            return info;
        }

        internal static DhcpClient ConvertFromNative(DHCP_CLIENT_INFO nativeClient)
        {
            DhcpClient client = new DhcpClient();
            client.IpAddress = new DhcpIpAddress(nativeClient.ClientIpAddress);
            client.SubnetMask = new DhcpIpAddress(nativeClient.SubnetMask);
            client.Name = nativeClient.ClientName;
            client.Comment = nativeClient.ClientComment;
            client.LeaseExpires = nativeClient.ClientLeaseExpires.Convert();
            Console.WriteLine(client.LeaseExpires.ToString());

            byte[] mac = new byte[nativeClient.ClientHardwareAddress.DataLength];
            Marshal.Copy(nativeClient.ClientHardwareAddress.Data, mac, 0, mac.Length);
            client.MacAddress = new DhcpMacAddress(mac);
            return client;
        }
    }

    public class DhcpSearchInfo
    {
        public DhcpSearchInfoType Type;
        public DhcpMacAddress MacAddress;
        public DhcpIpAddress IpAddress;
        public String Name;

        internal DHCP_SEARCH_INFO ConvertToNative(MemManager Mem)
        {
            DHCP_SEARCH_INFO info = new DHCP_SEARCH_INFO();
            switch (this.Type)
            {
                case DhcpSearchInfoType.IpAddress:
                    if (this.IpAddress == null)
                        throw new InvalidOperationException("SearchInfo invalid");

                    info.SearchType = DHCP_SEARCH_INFO_TYPE.DhcpClientIpAddress;
                    info.ClientIpAddress = this.IpAddress.GetUIntAddress();
                    return info;
                case DhcpSearchInfoType.HardwareAddress:
                    if (this.MacAddress == null)
                        throw new InvalidOperationException("SearchInfo invalid");
                    info.SearchType = DHCP_SEARCH_INFO_TYPE.DhcpClientHardwareAddress;

                    Byte[] mac = this.MacAddress.GetByteArray();
                    DHCP_CLIENT_UID uid = new DHCP_CLIENT_UID();
                    uid.DataLength = (uint)mac.Length;
                    uid.Data = Mem.AllocByteArray(this.MacAddress.GetByteArray());
                    info.ClientHardwareAddress = uid;

                    return info;
                case DhcpSearchInfoType.Name:
                    if (this.Name == null)
                        throw new InvalidOperationException("SearchInfo invalid");
                    info.SearchType = DHCP_SEARCH_INFO_TYPE.DhcpClientName;
                    info.ClientName = Mem.AllocString(this.Name);
                    return info;
            }
            return info;
        }
    }
    public enum DhcpSearchInfoType
    {
        IpAddress,
        HardwareAddress,
        Name
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpException.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;

using Unf.Dhcp.Smo.Interop;

namespace Unf.Dhcp.Smo
{
    public class DhcpException : ApplicationException
    {
        private uint DhcpErrCode;
        private bool isDhcpErr = true;

        public DhcpException(uint code)
        {
            DhcpErrCode = code;
            if(!Enum.IsDefined(typeof(DhcpsapiErrorType), code))
            { isDhcpErr = false; }
        }

        public override string Message
        {
            get
            {
                if (isDhcpErr)
                {
                    return DhcpsapiError.GetDesc((DhcpsapiErrorType)DhcpErrCode);
                }
                else
                {
                    Win32Exception ex = new Win32Exception((int)DhcpErrCode);
                    return ex.Message;
                }

            }
        }
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpIpAddress.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;

namespace Unf.Dhcp.Smo
{
    public class DhcpIpAddress
    {
        public DhcpIpAddress() { this.ip = IPAddress.Any.ToString(); }
        public DhcpIpAddress(String IpString) { this.Ip = IpString; }
        public DhcpIpAddress(UInt32 IpUInt) { this.Ip =
DhcpIpAddress.UInt2Str(IpUInt); }

        String ip;
        public String Ip
        {
            get
            {
                return ip;
            }
            set
            {
                IPAddress outIp;
                if (!IPAddress.TryParse(value, out outIp))
                {
                    ip = IPAddress.Any.ToString();
                    throw new FormatException("IpAddress invalid format");
                }
                ip = value;
            }
        }

        public override String ToString()
        {
            return Ip;
        }

        public UInt32 GetUIntAddress()
        {
            return DhcpIpAddress.Str2UInt(this.ip);
        }

        public static UInt32 Str2UInt(String IpString)
        {
            IPAddress ip = IPAddress.Parse(IpString);

            byte[] ipBytes = ip.GetAddressBytes();

            UInt32 ipUInt = (UInt32)ipBytes[0] << 24;
            ipUInt += (UInt32)ipBytes[1] << 16;
            ipUInt += (UInt32)ipBytes[2] << 8;
            ipUInt += (UInt32)ipBytes[3];

            return ipUInt;
        }

        public static String UInt2Str(UInt32 IpUInt)
        {
            IPAddress ip = new IPAddress(IpUInt);
            string[] strIp = ip.ToString().Split('.');

            return strIp[3] + "." + strIp[2] + "." + strIp[1] + "." + strIp[0];
        }
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpMacAddress.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;

using System.Text.RegularExpressions;
using System.Net.NetworkInformation;

namespace Unf.Dhcp.Smo
{
    public class DhcpMacAddress
    {

        public DhcpMacAddress() { this.mac = "000000000000"; }
        public DhcpMacAddress(String MacString) { this.Mac = MacString; }
        public DhcpMacAddress(Byte[] MacBytes)
        {
            PhysicalAddress InMac = new PhysicalAddress(MacBytes);
            this.mac = InMac.ToString();
        }

        private String mac;
        public String Mac
        {
            get { return this.mac; }
            set
            {
                String InMac = value.ToUpper();

                InMac = InMac.Replace(':', '-');

                this.mac = PhysicalAddress.Parse(InMac).ToString();
            }
        }

        public override String ToString()
        {
            return Mac;
        }

        public Byte[] GetByteArray()
        {
            return PhysicalAddress.Parse(this.mac).GetAddressBytes();
        }
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpOperations.asmx.cs
 *  Namespace: Unf.Dhcp.Smo.WebServices
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Data;
using System.Web;
using System.Collections;
using System.Collections.Generic;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.ComponentModel;
using System.Xml.Serialization;


using Unf.Dhcp.Smo;
using Unf.Dhcp.Smo.Operations;

namespace Unf.Dhcp.Smo.WebServices
{
    [WebService(Namespace = "http://www.unf.edu/~jrupard/dws/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class DhcpOperations : System.Web.Services.WebService
    {

        #region Subnet Ops...
        [WebMethod(Description = "Enumerate subnets on DHCP server")]
        public DhcpSubnet[] EnumSubnets(DhcpIpAddress server)
        {
            return AzServerOperations.EnumSubnets(server);
        }

        //secured
        [WebMethod(Description = "Create a DHCP subnet on server")]
        [XmlInclude(typeof(DhcpIpRange)), XmlInclude(typeof(BootpIpRange))]
        public void CreateSubnet(DhcpIpAddress server, DhcpSubnet network, DhcpIpRange
ipRange)
        {
            AzServerOperations.CreateSubnet(server, network, ipRange);
        }

        //secured
        [WebMethod(Description = "Update DHCP subnet on server")]
        public void UpdateSubnet(DhcpIpAddress server, DhcpSubnet network)
        {
            AzServerOperations.UpdateSubnet(server, network);
        }

        //secured
        [WebMethod(Description = "Update default DHCP subnet address pool (range)")]
        public void UpdateSubnetRange(DhcpIpAddress server, DhcpIpAddress network,
DhcpIpRange ipRange)
        {
            AzServerOperations.UpdateSubnetRange(server, network, ipRange);
        }

        //secured
        [WebMethod(Description = "Delete DHCP subnet on server")]
        public void DeleteSubnet(DhcpIpAddress server, DhcpIpAddress network)
        {
            AzServerOperations.DeleteSubnet(server, network);
        }

        //secured
        [WebMethod(Description = "Create a subnet exclusion address pool")]
        public void CreateSubnetExclusion(DhcpIpAddress server, DhcpIpAddress network,
DhcpIpRange ipExRange)
        {
            AzServerOperations.CreateSubnetExclusion(server, network, ipExRange);
        }

        //secured
        [WebMethod(Description = "Delete a subnet exclusion address pool")]
        public void DeleteSubnetExclusion(DhcpIpAddress server, DhcpIpAddress network,
DhcpIpRange ipExRange)
        {
            AzServerOperations.DeleteSubnetExclusion(server, network, ipExRange);
        }

        [WebMethod(Description = "Enumerate all address pools belonging to subnet")]
        public DhcpIpRange[] EnumPools(DhcpIpAddress server, DhcpIpAddress network)
        {
            return AzServerOperations.EnumPools(server, network);
        }
        //secured
        [WebMethod(Description = "Search subnet for client leases matching terms")]
        public DhcpClient[] SubnetFindAllClients(DhcpIpAddress server, DhcpIpAddress
network, DhcpClientFilter terms)
        {
            return AzServerOperations.SubnetFindAllClients(server, network, terms);
        }

        //secured
        [WebMethod(Description = "Search subnet for first client lease matching
terms")]
        public DhcpClient SubnetFindOneClient(DhcpIpAddress server, DhcpIpAddress
network, DhcpClientFilter terms)
        {
            return AzServerOperations.SubnetFindOneClient(server, network, terms);
        }

        //secured
        [WebMethod(Description = "Search <b>all</b> subnets on server looking for
client leases matching terms")]
        public DhcpClient[] ServerFindAllClients(DhcpIpAddress server,
DhcpClientFilter terms)
        {
            return AzServerOperations.ServerFindAllClients(server, terms);
        }

        //secured
        [WebMethod(Description = "Search <b>all</b> subnets on server looking for
first client lease matching terms")]
        public DhcpClient ServerFindOneClient(DhcpIpAddress server, DhcpClientFilter
terms)
        {
            return AzServerOperations.ServerFindOneClient(server, terms);
        }

#endregion

        #region Lease Ops...
        [WebMethod(Description = "Enumerate all client leases on subnet")]
        public DhcpClient[] EnumLeases(DhcpIpAddress server, DhcpIpAddress network)
        {
            return AzServerOperations.EnumLeases(server, network);
        }

        [WebMethod(Description = "Create a client lease on server")]
        public void CreateLease(DhcpIpAddress server, DhcpClient lease)
        {
            AzServerOperations.CreateLease(server, lease);
        }

        [WebMethod(Description = "Update a client lease on server")]
        public void UpdateLease(DhcpIpAddress server, DhcpClient lease)
        {
            AzServerOperations.UpdateLease(server, lease);
        }

        [WebMethod(Description = "Delete client lease from server")]
        public void DeleteLease(DhcpIpAddress server, DhcpIpAddress network,
DhcpIpAddress leaseIp)
        {
            AzServerOperations.DeleteLease(server, network, leaseIp);
        }
        #endregion

        #region Rez Ops...
        //requires ip+ create
        [WebMethod(Description = "Create a DHCP IP reservation on server")]
        public void CreateReservation(DhcpIpAddress server, DhcpIpAddress network,
DhcpReservation res)
```

```
        {
            AzServerOperations.CreateReservation(server, network, res);
        }

        [WebMethod(Description = "Update a DHCP IP reservation on server")]
        public void UpdateReservation(DhcpIpAddress server, DhcpIpAddress network,
DhcpReservation res)
        {
            AzServerOperations.UpdateReservation(server, network, res);
        }

        [WebMethod(Description = "Delete a DHCP IP reservation on server")]
        public void DeleteReservation(DhcpIpAddress server, DhcpIpAddress network,
DhcpIpAddress resIp)
        {
            AzServerOperations.DeleteReservation(server, network, resIp);
        }

        [WebMethod(Description = "Enumerate IP Reservations owned by subnet on
server")]
        public DhcpReservation[] EnumReservations(DhcpIpAddress server, DhcpIpAddress
network)
        {
            return AzServerOperations.EnumReservations(server, network);
        }

        #endregion

        #region Option Ops...
        [WebMethod(Description = "Set a DHCP option's value based on a particular
scope")]
        [XmlInclude(typeof(DhcpOptionDataByte)),
XmlInclude(typeof(DhcpOptionDataWord)), XmlInclude(typeof(DhcpOptionDataDword)),
        XmlInclude(typeof(DhcpOptionDataDwordDword)),
XmlInclude(typeof(DhcpOptionDataIp)), XmlInclude(typeof(DhcpOptionDataString))]
        public void SetOptionValue(DhcpIpAddress server, DhcpOptionValue value)
        {
            AzServerOperations.SetOptionValue(server, value);
        }

        [WebMethod(Description = "Remove a DHCP option's value from a particular
scope")]
        public void RemoveOptionValue(DhcpIpAddress server, DhcpOptionValue value)
        {
            AzServerOperations.RemoveOptionValue(server, value);
        }

        [WebMethod(Description = "Retrieve a DHCP option's value from a particular
scope")]
        public DhcpOptionValue GetOptionValue(DhcpIpAddress server, UInt32 optionId,
DhcpOptionClassType classType, DhcpOptionScope scope)
        {
            return AzServerOperations.GetOptionValue(server, optionId, classType,
scope);
        }

        [WebMethod(Description = "Enumerate all DHCP option values set in particular
scope")]
        public DhcpOptionValue[] EnumOptionValues(DhcpIpAddress server,
DhcpOptionClassType classType, DhcpOptionScope scope)
        {
            return AzServerOperations.EnumOptionValues(server, classType, scope);
        }

        [WebMethod(Description = "Enumerate all DHCP options available on server
(global set of DHCP options)")]
        public DhcpOption[] EnumOptions(DhcpIpAddress server, DhcpOptionClassType
classType)
        {
            return AzServerOperations.EnumOptions(server, classType);
        }
        #endregion

    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpOption.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;

using Unf.Dhcp.Smo.Interop;

namespace Unf.Dhcp.Smo
{
    public class DhcpOption
    {
        public UInt32 OptionId;
        public String Name;
        public String Comment;
        public DhcpOptionDataType DataType;
        public bool isArray;

        public DhcpOption() { }

        public static DhcpOption[] EnumAll(DhcpIpAddress server, DhcpOptionClassType
OptionClass)
        {
            UInt32 Response = 0, ResumeHandle = 0;
            UInt32 nRead = 0, nTotal = 0;
            IntPtr retPtr, iPtr;
            List<DhcpOption> options = new List<DhcpOption>();
            DHCP_OPTION nativeOption;
            DHCP_OPTION_ARRAY nativeArray;
            String ClassId = DhcpOptionClassHash.ht[OptionClass] as String;


            for ( ; ; )
            {
                Response = NativeMethods.DhcpEnumOptionsV5(server.ToString(), 0,
ClassId, null,
                    ref ResumeHandle, 65536, out retPtr, out nRead, out nTotal);

                //ERROR_NO_MORE_ITEMS
                if (Response == 259)
                    break;

                //ERROR_MORE_DATA = 234
                if (Response != 0 && Response != 234 || retPtr == IntPtr.Zero)
                {
                    if (retPtr != IntPtr.Zero)
                        NativeMethods.DhcpRpcFreeMemory(retPtr);
                    throw new DhcpException(Response);
                }

                //work
                DhcpOption opt;
                nativeArray = (DHCP_OPTION_ARRAY)Marshal.PtrToStructure(retPtr,
typeof(DHCP_OPTION_ARRAY));
```

```
                for (int i = 0; i < nativeArray.NumElements; ++i)
                {
                    iPtr = (IntPtr)(nativeArray.Elements.ToInt32() + (i *
Marshal.SizeOf(typeof(DHCP_OPTION))));
                    nativeOption = (DHCP_OPTION)Marshal.PtrToStructure(iPtr,
typeof(DHCP_OPTION));

                    opt = DhcpOption.ConvertFromNative(nativeOption);
                    if(opt != null)
                        options.Add(opt);
                }

                //free on last successful call
                if (Response == 0)
                    NativeMethods.DhcpRpcFreeMemory(retPtr);
            }

            return options.ToArray();
        }

        internal static DhcpOption ConvertFromNative(DHCP_OPTION nativeOption)
        {
            DhcpOption option = new DhcpOption();

            //unsupported options
            DHCP_OPTION_DATA_ELEMENT element =
(DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(
               nativeOption.DefaultValue.Elements, typeof(DHCP_OPTION_DATA_ELEMENT));

            if(element.OptionType == DHCP_OPTION_DATA_TYPE.DhcpBinaryDataOption ||
                element.OptionType == DHCP_OPTION_DATA_TYPE.DhcpEncapsulatedDataOption
||
                element.OptionType == DHCP_OPTION_DATA_TYPE.DhcpIpv6AddressOption)
                    return null;

            option.DataType = (DhcpOptionDataType)element.OptionType;
            option.OptionId = nativeOption.OptionID;
            option.Name = nativeOption.OptionName;
            option.Comment = nativeOption.OptionComment;
            option.isArray = nativeOption.OptionType ==
DHCP_OPTION_TYPE.DhcpArrayTypeOption ? true : false;

            return option;
        }
    }

    public enum DhcpOptionClassType
    {
        Dhcp,
        Bootp,
        Routing
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpOptionValue.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;

using System.Runtime.InteropServices;
using Unf.Dhcp.Smo.Interop;

namespace Unf.Dhcp.Smo
{

    public class DhcpOptionValue
    {
        public UInt32 OptionId;
        public DhcpOptionClassType ClassType;
        public DhcpOptionData OptionData;
        public DhcpOptionScope OptionScope;

        public DhcpOptionValue()
        {
            this.OptionId = UInt32.MaxValue;
            this.ClassType = DhcpOptionClassType.Dhcp;
        }

        public void Set(DhcpIpAddress Server)
        {
            if (this.OptionId == UInt32.MaxValue || this.OptionScope == null)
                throw new InvalidOperationException("Invalid Option Data");

            //Run Check to make sure Id exist and types (data, isArray) are correct
            //DhcpOption.ValidateOptionValue(Server, this);

            //Get Option Class
            String ClassId = DhcpOptionClassHash.ht[this.ClassType] as String;

            //Get Scope Setting
            DHCP_OPTION_SCOPE_INFO NativeScopeInfo =
this.OptionScope.ConvertToNative();

            uint Response = 0;
            using (MemManager Mem = new MemManager())
            {
                DHCP_OPTION_DATA NativeOptionData =
this.OptionData.ConvertToNative(Mem);

                Response = NativeMethods.DhcpSetOptionValueV5(Server.ToString(), 0,
this.OptionId, ClassId, null, ref NativeScopeInfo, ref NativeOptionData);
            }

            if (Response != 0)
                throw new DhcpException(Response);

        }

        public void Remove(DhcpIpAddress Server)
        {
            if (this.OptionId == UInt32.MaxValue || this.OptionScope == null)
                throw new InvalidOperationException("Invalid Option Data");

            String ClassId = DhcpOptionClassHash.ht[this.ClassType] as String;

            //Get Scope Setting
            DHCP_OPTION_SCOPE_INFO NativeScopeInfo =
this.OptionScope.ConvertToNative();

            uint Response = NativeMethods.DhcpRemoveOptionValueV5(Server.ToString(),
0, this.OptionId, ClassId, null, ref NativeScopeInfo);

            if (Response != 0)
                throw new DhcpException(Response);
        }

        public void Get(DhcpIpAddress Server)
        {
            if (this.OptionId == UInt32.MaxValue || this.OptionScope == null)
```

```
                throw new InvalidOperationException("Invalid Option Data");

            String ClassId = DhcpOptionClassHash.ht[this.ClassType] as String;

            //Get Scope Setting
            DHCP_OPTION_SCOPE_INFO NativeScopeInfo =
this.OptionScope.ConvertToNative();

            IntPtr iPtr = IntPtr.Zero;
            uint Response = NativeMethods.DhcpGetOptionValueV5(Server.ToString(), 0,
this.OptionId, ClassId, null, ref NativeScopeInfo, out iPtr);

            if (Response != 0)
            {
                if (iPtr != IntPtr.Zero)
                    NativeMethods.DhcpRpcFreeMemory(iPtr);

                throw new DhcpException(Response);
            }

            //
            DHCP_OPTION_VALUE NativeOptVal =
(DHCP_OPTION_VALUE)Marshal.PtrToStructure(iPtr, typeof(DHCP_OPTION_VALUE));

            this.OptionData = DhcpOptionData.CovertFromNative(NativeOptVal.Value);


        }

        public static DhcpOptionValue[] EnumAll(DhcpIpAddress Server,
DhcpOptionClassType OptionClass, DhcpOptionScope Scope)
        {
            DHCP_OPTION_VALUE NativeOptionValue;
            IntPtr retPtr, iPtr;
            DHCP_OPTION_VALUE_ARRAY NativeOptVArray;
            List<DhcpOptionValue> ovList = new List<DhcpOptionValue>();
            DhcpOptionValue  ov;

            //Get Scope Setting
            DHCP_OPTION_SCOPE_INFO NativeScopeInfo = Scope.ConvertToNative();

            String ClassId = DhcpOptionClassHash.ht[OptionClass] as String;

            UInt32 rHandle = 0, OptionsRead = 0, OptionsTotal = 0, Response = 0;
            for ( ; ; )
            {
                Response = NativeMethods.DhcpEnumOptionValuesV5(Server.ToString(), 0,
ClassId, null, ref NativeScopeInfo, ref rHandle,
                    65536, out retPtr, out OptionsRead, out OptionsTotal);

                //ERROR_NO_MORE_ITEMS
                if (Response == 259)
                    break;

                //ERROR_MORE_DATA = 234
                if(Response != 0 && Response != 234 || retPtr == IntPtr.Zero)
                {
                    if(retPtr != IntPtr.Zero)
                        NativeMethods.DhcpRpcFreeMemory(retPtr);
                    throw new DhcpException(Response);
                }

                NativeOptVArray =
(DHCP_OPTION_VALUE_ARRAY)Marshal.PtrToStructure(retPtr,
typeof(DHCP_OPTION_VALUE_ARRAY));
                for (int i = 0; i < NativeOptVArray.NumElements; ++i)
                {
                    iPtr = (IntPtr)(NativeOptVArray.Elements.ToInt32() + (i *
Marshal.SizeOf(typeof(DHCP_OPTION_VALUE))));
```

```
                    NativeOptionValue =
(DHCP_OPTION_VALUE)Marshal.PtrToStructure(iPtr, typeof(DHCP_OPTION_VALUE));

                    ov = new DhcpOptionValue();
                    ov.ClassType = OptionClass;
                    ov.OptionScope = Scope;
                    ov.OptionId = NativeOptionValue.OptionID;
                    ov.OptionData =
DhcpOptionData.CovertFromNative(NativeOptionValue.Value);

                    ovList.Add(ov);

                }

                //free on last successful call
                if (Response == 0)
                    NativeMethods.DhcpRpcFreeMemory(retPtr);
            }

            return ovList.ToArray();
        }
    }

    public class DhcpOptionScope
    {
        public DhcpOptionScopeType type;
        public DhcpIpAddress SubnetIp;
        public DhcpIpAddress ReservationIp;

        public DhcpOptionScope()
        {
            SubnetIp = null;
            ReservationIp = null;
        }

        internal DHCP_OPTION_SCOPE_INFO ConvertToNative()
        {
            DHCP_OPTION_SCOPE_INFO sInfo = new DHCP_OPTION_SCOPE_INFO();
            switch (this.type)
            {
                case DhcpOptionScopeType.Server:
                    sInfo.ScopeType = DHCP_OPTION_SCOPE_TYPE.DhcpGlobalOptions;
                    sInfo.GlobalScopeInfo = IntPtr.Zero;
                    break;
                case DhcpOptionScopeType.Subnet:
                    sInfo.ScopeType = DHCP_OPTION_SCOPE_TYPE.DhcpSubnetOptions;
                    if (this.SubnetIp == null)
                        throw new InvalidOperationException("Subnet IP not assigned");
                    sInfo.SubnetScopeInfo = this.SubnetIp.GetUIntAddress();
                    break;
                case DhcpOptionScopeType.Reservation:
                    sInfo.ScopeType = DHCP_OPTION_SCOPE_TYPE.DhcpReservedOptions;
                    if (this.SubnetIp == null)
                        throw new InvalidOperationException("Subnet IP not assigned");
                    if (this.ReservationIp == null)
                        throw new InvalidOperationException("Reservation IP not
assigned");
                    sInfo.ReservedScopeInfo = new DHCP_RESERVED_SCOPE();
                    sInfo.ReservedScopeInfo.ReservedIpSubnetAddress =
this.SubnetIp.GetUIntAddress();
                    sInfo.ReservedScopeInfo.ReservedIpAddress =
this.ReservationIp.GetUIntAddress();
                    break;
            }
            return sInfo;
        }
    }

    #region OptionData...
    public abstract class DhcpOptionData
    {
        public DhcpOptionDataType Type;
        public DhcpOptionData() { }

        public DhcpOptionData(DhcpOptionDataType type)
        {

            this.Type = type;
        }

        internal abstract DHCP_OPTION_DATA ConvertToNative(MemManager Mem);

        internal static DhcpOptionData CovertFromNative(DHCP_OPTION_DATA
NativeOptData)
        {
            if (NativeOptData.NumElements == 0 || NativeOptData.Elements ==
IntPtr.Zero)
                throw new ArgumentException("DHCP_OPTION_DATA not valid");

            //grab first element for type test
            DHCP_OPTION_DATA_ELEMENT element =
(DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(NativeOptData.Elements,
typeof(DHCP_OPTION_DATA_ELEMENT));
            switch (element.OptionType)
            {
                case DHCP_OPTION_DATA_TYPE.DhcpByteOption:
                    return new DhcpOptionDataByte(NativeOptData);
                case DHCP_OPTION_DATA_TYPE.DhcpDWordDWordOption:
                    return new DhcpOptionDataDwordDword(NativeOptData);
                case DHCP_OPTION_DATA_TYPE.DhcpDWordOption:
                    return new DhcpOptionDataDword(NativeOptData);
                case DHCP_OPTION_DATA_TYPE.DhcpIpAddressOption:
                    return new DhcpOptionDataIp(NativeOptData);
                case DHCP_OPTION_DATA_TYPE.DhcpStringDataOption:
                    return new DhcpOptionDataString(NativeOptData);
                case DHCP_OPTION_DATA_TYPE.DhcpWordOption:
                    return new DhcpOptionDataWord(NativeOptData);
                default:
                    throw new NotImplementedException("OptionData Type: " +
element.OptionType);
            }
        }

    }

    public class DhcpOptionDataByte : DhcpOptionData
    {
        public Byte[] Data;
        public DhcpOptionDataByte() { this.Type = DhcpOptionDataType.ByteType; }
        public DhcpOptionDataByte(Byte[] bytes)
            : base(DhcpOptionDataType.ByteType)
        {
            this.Data = bytes;
        }

        internal DhcpOptionDataByte(DHCP_OPTION_DATA NativeOptData)
        {
            if (NativeOptData.NumElements == 0 || NativeOptData.Elements ==
IntPtr.Zero)
                throw new ArgumentException("DHCP_OPTION_DATA not valid");

            this.Type = DhcpOptionDataType.ByteType;

            //grab first element for type test
            DHCP_OPTION_DATA_ELEMENT element =
(DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(NativeOptData.Elements,
typeof(DHCP_OPTION_DATA_ELEMENT));
```

```
            if (element.OptionType != DHCP_OPTION_DATA_TYPE.DhcpByteOption)
                throw new ArgumentException("NativeOptData type not Byte");

            this.Data = new Byte[NativeOptData.NumElements];

            this.Data[0] = element.ByteOption;

            IntPtr iPtr;
            for (int i = 1; i < NativeOptData.NumElements; ++i)
            {
                iPtr = (IntPtr)(NativeOptData.Elements.ToInt32() + (i *
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                element = (DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(iPtr,
typeof(DHCP_OPTION_DATA_ELEMENT));
                this.Data[i] = element.ByteOption;
            }
        }

        internal override DHCP_OPTION_DATA ConvertToNative(MemManager Mem)
        {

            if (this.Data.Length == 0)
                throw new InvalidOperationException("OptionValue: No data to
convert");

            DHCP_OPTION_DATA_ELEMENT element;
            DHCP_OPTION_DATA data = new DHCP_OPTION_DATA();

            data.NumElements = (UInt32)this.Data.Length;
            data.Elements = Mem.AllocArray(typeof(DHCP_OPTION_DATA_ELEMENT),
data.NumElements);
            IntPtr iptr;
            for (int i = 0; i < data.NumElements; i++)
            {
                element = new DHCP_OPTION_DATA_ELEMENT();
                element.OptionType = DHCP_OPTION_DATA_TYPE.DhcpByteOption;
                element.ByteOption = this.Data[i];

                iptr = (IntPtr)(data.Elements.ToInt32() + (i *
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                Marshal.StructureToPtr(element, iptr, false);

            }
            return data;
        }
    }

    public class DhcpOptionDataIp : DhcpOptionData
    {
        public DhcpIpAddress[] Data;
        public DhcpOptionDataIp() { this.Type = DhcpOptionDataType.IpAddressType; }
        public DhcpOptionDataIp(DhcpIpAddress[] ips)
            : base(DhcpOptionDataType.IpAddressType)
        {
            this.Data = ips;
        }

        internal DhcpOptionDataIp(DHCP_OPTION_DATA NativeOptData)
        {
            if (NativeOptData.NumElements == 0 || NativeOptData.Elements ==
IntPtr.Zero)
                throw new ArgumentException("DHCP_OPTION_DATA not valid");

            this.Type = DhcpOptionDataType.IpAddressType;

            //grab first element for type test
            DHCP_OPTION_DATA_ELEMENT element =
(DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(NativeOptData.Elements,
typeof(DHCP_OPTION_DATA_ELEMENT));
```

```
            if (element.OptionType != DHCP_OPTION_DATA_TYPE.DhcpIpAddressOption)
                throw new ArgumentException("NativeOptData type not IP");

            this.Data = new DhcpIpAddress[NativeOptData.NumElements];

            this.Data[0] = new DhcpIpAddress(element.DWordOption);

            IntPtr iPtr;
            for (int i = 1; i < NativeOptData.NumElements; ++i)
            {
                iPtr = (IntPtr)(NativeOptData.Elements.ToInt32() + (i * 
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                element = (DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(iPtr, 
typeof(DHCP_OPTION_DATA_ELEMENT));
                this.Data[i] = new DhcpIpAddress(element.DWordOption);
            }
        }

        internal override DHCP_OPTION_DATA ConvertToNative(MemManager Mem)
        {

            if (this.Data.Length == 0)
                throw new InvalidOperationException("OptionValue: No data to 
convert");

            DHCP_OPTION_DATA_ELEMENT element;
            DHCP_OPTION_DATA data = new DHCP_OPTION_DATA();

            data.NumElements = (UInt32)this.Data.Length;
            data.Elements = Mem.AllocArray(typeof(DHCP_OPTION_DATA_ELEMENT), 
data.NumElements);
            IntPtr iptr;
            for (int i = 0; i < data.NumElements; i++)
            {
                element = new DHCP_OPTION_DATA_ELEMENT();
                element.OptionType = DHCP_OPTION_DATA_TYPE.DhcpIpAddressOption;
                element.DWordOption = this.Data[i].GetUIntAddress();

                iptr = (IntPtr)(data.Elements.ToInt32() + (i * 
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                Marshal.StructureToPtr(element, iptr, false);

            }
            return data;
        }
    }

    public class DhcpOptionDataWord : DhcpOptionData
    {
        public UInt16[] Data;
        public DhcpOptionDataWord() { this.Type = DhcpOptionDataType.WordType; }
        public DhcpOptionDataWord(UInt16[] words)
            : base(DhcpOptionDataType.WordType)
        {
            this.Data = words;
        }

        internal DhcpOptionDataWord(DHCP_OPTION_DATA NativeOptData)
        {
            if (NativeOptData.NumElements == 0 || NativeOptData.Elements == 
IntPtr.Zero)
                throw new ArgumentException("DHCP_OPTION_DATA not valid");

            this.Type = DhcpOptionDataType.WordType;

            //grab first element for type test
            DHCP_OPTION_DATA_ELEMENT element =
(DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(NativeOptData.Elements,
typeof(DHCP_OPTION_DATA_ELEMENT));

            if (element.OptionType != DHCP_OPTION_DATA_TYPE.DhcpWordOption)
                throw new ArgumentException("NativeOptData type not WORD");

            this.Data = new UInt16[NativeOptData.NumElements];

            this.Data[0] = element.WordOption;

            IntPtr iPtr;
            for (int i = 1; i < NativeOptData.NumElements; ++i)
            {
                iPtr = (IntPtr)(NativeOptData.Elements.ToInt32() + (i *
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                element = (DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(iPtr,
typeof(DHCP_OPTION_DATA_ELEMENT));
                this.Data[i] = element.WordOption;
            }
        }

        internal override DHCP_OPTION_DATA ConvertToNative(MemManager Mem)
        {

            if (this.Data.Length == 0)
                throw new InvalidOperationException("OptionValue: No data to
convert");

            DHCP_OPTION_DATA_ELEMENT element;
            DHCP_OPTION_DATA data = new DHCP_OPTION_DATA();

            data.NumElements = (UInt32)this.Data.Length;
            data.Elements = Mem.AllocArray(typeof(DHCP_OPTION_DATA_ELEMENT),
data.NumElements);
            IntPtr iptr;
            for (int i = 0; i < data.NumElements; i++)
            {
                element = new DHCP_OPTION_DATA_ELEMENT();
                element.OptionType = DHCP_OPTION_DATA_TYPE.DhcpWordOption;
                element.WordOption = this.Data[i];

                iptr = (IntPtr)(data.Elements.ToInt32() + (i *
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                Marshal.StructureToPtr(element, iptr, false);

            }
            return data;
        }
    }

    public class DhcpOptionDataDword : DhcpOptionData
    {
        public UInt32[] Data;
        public DhcpOptionDataDword() { this.Type = DhcpOptionDataType.DWordType; }
        public DhcpOptionDataDword(UInt32[] dwords)
            : base(DhcpOptionDataType.DWordType)
        {
            this.Data = dwords;
        }

        internal DhcpOptionDataDword(DHCP_OPTION_DATA NativeOptData)
        {
            if (NativeOptData.NumElements == 0 || NativeOptData.Elements ==
IntPtr.Zero)
                throw new ArgumentException("DHCP_OPTION_DATA not valid");

            this.Type = DhcpOptionDataType.DWordType;
            //grab first element for type test
            DHCP_OPTION_DATA_ELEMENT element = 
(DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(NativeOptData.Elements, 
typeof(DHCP_OPTION_DATA_ELEMENT));

            if (element.OptionType != DHCP_OPTION_DATA_TYPE.DhcpDWordOption)
                throw new ArgumentException("NativeOptData type not DWORD");

            this.Data = new UInt32[NativeOptData.NumElements];

            this.Data[0] = element.DWordOption;

            IntPtr iPtr;
            for (int i = 1; i < NativeOptData.NumElements; ++i)
            {
                iPtr = (IntPtr)(NativeOptData.Elements.ToInt32() + (i * 
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                element = (DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(iPtr, 
typeof(DHCP_OPTION_DATA_ELEMENT));
                this.Data[i] = element.DWordOption;
            }
        }

        internal override DHCP_OPTION_DATA ConvertToNative(MemManager Mem)
        {

            if (this.Data.Length == 0)
                throw new InvalidOperationException("OptionValue: No data to 
convert");

            DHCP_OPTION_DATA_ELEMENT element;
            DHCP_OPTION_DATA data = new DHCP_OPTION_DATA();

            data.NumElements = (UInt32)this.Data.Length;
            data.Elements = Mem.AllocArray(typeof(DHCP_OPTION_DATA_ELEMENT), 
data.NumElements);
            IntPtr iptr;
            for (int i = 0; i < data.NumElements; i++)
            {
                element = new DHCP_OPTION_DATA_ELEMENT();
                element.OptionType = DHCP_OPTION_DATA_TYPE.DhcpDWordOption;
                element.DWordOption = this.Data[i];

                iptr = (IntPtr)(data.Elements.ToInt32() + (i * 
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                Marshal.StructureToPtr(element, iptr, false);

            }
            return data;
        }
    }

    public class DhcpOptionDataDwordDword : DhcpOptionData
    {
        public UInt64[] Data;
        public DhcpOptionDataDwordDword() { this.Type = 
DhcpOptionDataType.DWordDWordType; }
        public DhcpOptionDataDwordDword(UInt64[] ddwords)
            : base(DhcpOptionDataType.DWordDWordType)
        {
            this.Data = ddwords;
        }

        internal DhcpOptionDataDwordDword(DHCP_OPTION_DATA NativeOptData)
        {
            if (NativeOptData.NumElements == 0 || NativeOptData.Elements == 
IntPtr.Zero)
                throw new ArgumentException("DHCP_OPTION_DATA not valid");
            this.Type = DhcpOptionDataType.DWordDWordType;

            //grab first element for type test
            DHCP_OPTION_DATA_ELEMENT element = 
(DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(NativeOptData.Elements, 
typeof(DHCP_OPTION_DATA_ELEMENT));

            if (element.OptionType != DHCP_OPTION_DATA_TYPE.DhcpDWordDWordOption)
                throw new ArgumentException("NativeOptData type not DWORDDWORD");

            this.Data = new UInt64[NativeOptData.NumElements];

            this.Data[0] = ((((UInt64)element.DWordDWordOption.UpperWord1) << 32) | 
element.DWordDWordOption.LowerWord2);

            IntPtr iPtr;
            for (int i = 1; i < NativeOptData.NumElements; ++i)
            {
                iPtr = (IntPtr)(NativeOptData.Elements.ToInt32() + (i * 
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                element = (DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(iPtr, 
typeof(DHCP_OPTION_DATA_ELEMENT));
                this.Data[i] = ((((UInt64)element.DWordDWordOption.UpperWord1) << 32) 
| element.DWordDWordOption.LowerWord2); ;
            }
        }

        internal override DHCP_OPTION_DATA ConvertToNative(MemManager Mem)
        {

            if (this.Data.Length == 0)
                throw new InvalidOperationException("OptionValue: No data to 
convert");

            DHCP_OPTION_DATA_ELEMENT element;
            DHCP_OPTION_DATA data = new DHCP_OPTION_DATA();

            data.NumElements = (UInt32)this.Data.Length;
            data.Elements = Mem.AllocArray(typeof(DHCP_OPTION_DATA_ELEMENT), 
data.NumElements);
            IntPtr iptr;
            for (int i = 0; i < data.NumElements; i++)
            {
                element = new DHCP_OPTION_DATA_ELEMENT();
                element.OptionType = DHCP_OPTION_DATA_TYPE.DhcpDWordDWordOption;
                element.DWordDWordOption.LowerWord2 = (UInt32)(this.Data[i] & 
0xFFFFFFFF);
                element.DWordDWordOption.UpperWord1 = (UInt32)((this.Data[i] & 
0xFFFFFFFF00000000) >> 32);

                iptr = (IntPtr)(data.Elements.ToInt32() + (i * 
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                Marshal.StructureToPtr(element, iptr, false);

            }
            return data;
        }
    }

    public class DhcpOptionDataString : DhcpOptionData
    {
        public String[] Data;
        public DhcpOptionDataString() { this.Type = DhcpOptionDataType.StringDataType; 
}
        public DhcpOptionDataString(String[] strings)
            : base(DhcpOptionDataType.StringDataType)
        {
            this.Data = strings;
        }
        internal DhcpOptionDataString(DHCP_OPTION_DATA NativeOptData)
        {
            if (NativeOptData.NumElements == 0 || NativeOptData.Elements ==
IntPtr.Zero)
                throw new ArgumentException("DHCP_OPTION_DATA not valid");

            this.Type = DhcpOptionDataType.StringDataType;

            //grab first element for type test
            DHCP_OPTION_DATA_ELEMENT element =
(DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(NativeOptData.Elements,
typeof(DHCP_OPTION_DATA_ELEMENT));

            if (element.OptionType != DHCP_OPTION_DATA_TYPE.DhcpStringDataOption)
                throw new ArgumentException("NativeOptData type not String");

            this.Data = new String[NativeOptData.NumElements];

            this.Data[0] = Marshal.PtrToStringAuto(element.StringOption);

            IntPtr iPtr;
            for (int i = 1; i < NativeOptData.NumElements; ++i)
            {
                iPtr = (IntPtr)(NativeOptData.Elements.ToInt32() + (i *
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                element = (DHCP_OPTION_DATA_ELEMENT)Marshal.PtrToStructure(iPtr,
typeof(DHCP_OPTION_DATA_ELEMENT));
                this.Data[i] = Marshal.PtrToStringAuto(element.StringOption);
            }
        }

        internal override DHCP_OPTION_DATA ConvertToNative(MemManager Mem)
        {

            if (this.Data.Length == 0)
                throw new InvalidOperationException("OptionValue: No data to
convert");

            DHCP_OPTION_DATA_ELEMENT element;
            DHCP_OPTION_DATA data = new DHCP_OPTION_DATA();

            data.NumElements = (UInt32)this.Data.Length;
            data.Elements = Mem.AllocArray(typeof(DHCP_OPTION_DATA_ELEMENT),
data.NumElements);
            IntPtr iptr;
            for (int i = 0; i < data.NumElements; i++)
            {
                element = new DHCP_OPTION_DATA_ELEMENT();
                element.OptionType = DHCP_OPTION_DATA_TYPE.DhcpStringDataOption;
                element.StringOption = Mem.AllocString(this.Data[i]);

                iptr = (IntPtr)(data.Elements.ToInt32() + (i *
Marshal.SizeOf(typeof(DHCP_OPTION_DATA_ELEMENT))));
                Marshal.StructureToPtr(element, iptr, false);

            }
            return data;
        }
    }
    #endregion

    static internal class DhcpOptionClassHash
    {
        public static Hashtable ht = new Hashtable();
        static DhcpOptionClassHash()
        {
            ht.Add(DhcpOptionClassType.Dhcp, null);
            ht.Add(DhcpOptionClassType.Bootp, "Default BOOTP Class");
            ht.Add(DhcpOptionClassType.Routing, "Default Routing and Remote Access
Class");
        }
    }

    #region Enums...
    public enum DhcpOptionDataType
    {
        ByteType,
        WordType,
        DWordType,
        DWordDWordType,
        IpAddressType,
        StringDataType
     //   BinaryDataType,
     //   EncapsulatedDataType,
     //   Ipv6AddressType
    }

    public enum DhcpOptionScopeType
    {
        Server,
        Subnet,
        Reservation
    }
    #endregion
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpReservation.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;


namespace Unf.Dhcp.Smo
{
    public class DhcpReservation
    {
        public DhcpIpAddress ReservedIp;
        public DhcpMacAddress ReservedMac;
        public DhcpSubnetClientType bAllowedClientTypes;
        public String Name;
        public String Comment;

        public DhcpReservation() {}
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpSearch.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;
using System.Text.RegularExpressions;

namespace Unf.Dhcp.Smo
```

```
{
    public class DhcpClientFilter
    {
        private Regex ipRegEx;
        public String IpRegEx
        {
            get { return ipRegEx.ToString(); }
            set
            {
                if (String.IsNullOrEmpty(value))
                    ipRegEx = null;
                else
                    ipRegEx = new Regex(value, RegexOptions.IgnoreCase |
RegexOptions.Compiled);

            }
        }

        private Regex macRegEx;
        public String MacRegEx
        {
            get { return macRegEx.ToString(); }
            set
            {
                if (String.IsNullOrEmpty(value))
                    macRegEx = null;
                else
                    macRegEx = new Regex(value, RegexOptions.IgnoreCase |
RegexOptions.Compiled);

            }
        }

        private Regex nameRegEx;
        public String NameRegEx
        {
            get { return nameRegEx.ToString(); }
            set
            {
                if (String.IsNullOrEmpty(value))
                    nameRegEx = null;
                else
                    nameRegEx = new Regex(value, RegexOptions.IgnoreCase |
RegexOptions.Compiled);
            }
        }

        public DhcpClientFilter() { }

        public bool Filter(DhcpClient c)
        {
            if (nameRegEx == null && ipRegEx == null && macRegEx == null)
                return false;

            if ((nameRegEx == null || nameRegEx.IsMatch(c.Name)) &&
                (ipRegEx == null || ipRegEx.IsMatch(c.IpAddress.ToString())) &&
                (macRegEx == null || macRegEx.IsMatch(c.MacAddress.ToString())))
                return true;
            return false;
        }
    }

}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpSecurity.asmx.cs
 *  Namespace: Unf.Dhcp.Smo.WebServices
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Data;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.ComponentModel;
using System.Xml.Serialization;
using Microsoft.Interop.Security.AzRoles;

namespace Unf.Dhcp.Smo.WebServices
{
    [WebService(Namespace = "http://www.unf.edu/~jrupard/dws/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class DhcpSecurity : System.Web.Services.WebService
    {

        #region Perms...

        [WebMethod(Description = "Add account with role to DHCP authorization 
database")]
        [XmlInclude(typeof(DhcpAzServerScope)), XmlInclude(typeof(DhcpAzSubnetScope)), 
XmlInclude(typeof(DhcpAzIpRangeScope))]
        public void AddPermission(DhcpAzScope scope, DhcpAzAccount account)
        {
            if (!DhcpSecurityOps.isAdmin())
                throw new UnauthorizedAccessException(AdminAccessErr);
            DhcpSecurityOps.AddPermission(scope, account.Role, account.AccountName);
        }

        [WebMethod(Description = "Remove account with role from DHCP authorization 
database")]
        public void RemovePermission(DhcpAzScope scope, DhcpAzAccount account)
        {
            if (!DhcpSecurityOps.isAdmin())
                throw new UnauthorizedAccessException(AdminAccessErr);
            DhcpSecurityOps.RemovePermission(scope, account.Role, 
account.AccountName);
        }

        [WebMethod(Description = "Enumerate accounts and their roles in DHCP 
authorization database")]
        public DhcpAzAccount[] GetPermissions(DhcpAzScope scope)
        {
            if (!DhcpSecurityOps.isAdmin())
                throw new UnauthorizedAccessException(AdminAccessErr);
            return DhcpSecurityOps.GetPermissions(scope);
        }

        [WebMethod(Description = "Add account with global role to DHCP authorization 
database")]
        public void AddGlobalPermission(DhcpAzGlobalAccount account)
        {
            if (!DhcpSecurityOps.isAdmin())
                throw new UnauthorizedAccessException(AdminAccessErr);
            DhcpSecurityOps.AddGlobalPermission(account.GlobalRole, 
account.AccountName);
        }

        [WebMethod(Description = "Enumerate global accounts and their roles in DHCP 
authorization database")]
        public DhcpAzGlobalAccount[] GetGlobalPermissions()
        {
            if (!DhcpSecurityOps.isAdmin())
                throw new UnauthorizedAccessException(AdminAccessErr);
            return DhcpSecurityOps.GetGlobalPermissions();
        }

        [WebMethod(Description = "Remove account with global role from DHCP
authorization database")]
        public void RemoveGlobalPermission(DhcpAzGlobalAccount account)
        {
            if (!DhcpSecurityOps.isAdmin())
                throw new UnauthorizedAccessException(AdminAccessErr);

            DhcpSecurityOps.RemoveGlobalPermission(account.GlobalRole,
account.AccountName);
        }

        [WebMethod(Description = "Enumerate authorization IP ranges in DHCP
authorization database")]
        public DhcpAzIpRangeScope[] EnumAzIpRanges(DhcpAzSubnetScope scope)
        {
            if (!DhcpSecurityOps.isAdmin())
                throw new UnauthorizedAccessException(AdminAccessErr);

            return DhcpSecurityOps.EnumAzIpRanges(scope);
        }

        [WebMethod(Description = "Check if current user belongs to global
administrator role in DHCP authorization database")]
        public Boolean isAdmin()
        {
            return DhcpSecurityOps.isAdmin();
        }
        #endregion

        #region Const...
        private const String AdminAccessErr = "Must be in the Global Administrator
Role or BUILTIN\\Administrators Local Server Group";
        #endregion
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpSecurityOps.cs
 *  Namespace: Unf.Dhcp.Smo.WebServices
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Configuration;
using System.Collections.Generic;
using System.Web;
using System.Web.Security;
using System.Text.RegularExpressions;

using System.Runtime.InteropServices;
using Microsoft.Interop.Security.AzRoles;
using System.Security.Principal;

namespace Unf.Dhcp.Smo.WebServices
{
    public class DhcpSecurityOps
    {
        public static void Init()
        {
            IAzApplication2 AzManApp;
            AzAuthorizationStoreClass AzManStore;
            AzManStore = new AzAuthorizationStoreClass();
            AzManStore.Initialize(0,
ConfigurationManager.AppSettings["AZ_STORE_PATH"], null);
            AzManApp =
AzManStore.OpenApplication2(ConfigurationManager.AppSettings["AZ_APP_NAME"], null);

            HttpContext.Current.Application["AZ_STORE_OBJ"] = AzManStore;
            HttpContext.Current.Application["AZ_APP_OBJ"] = AzManApp;

        }

        private static IAzApplication2 GetAzApplication()
        {
            return (IAzApplication2)HttpContext.Current.Application["AZ_APP_OBJ"];
        }

        public static IAzClientContext2 GetClientContext()
        {
            WindowsIdentity wid = (WindowsIdentity)HttpContext.Current.User.Identity;
            IntPtr tokenPtr = wid.Token;
            return
(IAzClientContext2)GetAzApplication().InitializeClientContextFromToken((ulong)tokenPtr
, 0);
        }

        public static Boolean isAdmin()
        {
            return HttpContext.Current.User.IsInRole(@"BUILTIN\Administrators") ||
isAzGlobalAdmin();
        }

        private static Boolean isAzGlobalAdmin()
        {
            IAzClientContext2 ctx = GetClientContext();
            Object[] objRoles = (Object[])ctx.GetRoles("");
            String GlobalAdminRole = Enum.GetName(typeof(DhcpAzGlobalRole),
DhcpAzGlobalRole.GlobalAdministrator);

            foreach (String role in objRoles)
                if (String.Compare(GlobalAdminRole, role) == 0)
                    return true;

            return false;
        }

        private static IAzScope GetAzScope(String scope)
        {
            try
            {
                return GetAzApplication().OpenScope(scope, null);
            }
            catch
            {
                return null;
            }
        }

        private static IAzScope CreateCustomAzScope(String scope)
        {
            IAzApplication2 azApp = GetAzApplication();
            IAzScope azScope = azApp.CreateScope(scope, null);
            azScope.Submit(0, null);

            IAzRole role = null;
            foreach (String item in Enum.GetNames(typeof(DhcpAzRole)))
            {
                try
                {
                    //create role in scope
                    role = azScope.CreateRole(item, null);
                    //assign global role definition (role defs are task objects
                    role.AddTask(item, null);
                    role.Submit(0, null);
                }
                finally
                {
                    FreeCom(role, false);
                }
            }

            //create special 'Limited' role
            try
            {

                role = azScope.CreateRole("Limited", null);
                //assign global role definition (role defs are task objects
                role.AddTask("Limited", null);
                role.Submit(0, null);
            }
            finally
            {
                FreeCom(role, false);
            }

            return azScope;
        }

        public static DhcpAzIpRangeScope[] EnumAzIpRanges(DhcpAzSubnetScope
azSubnetScope)
        {
            List<DhcpAzIpRangeScope> ranges = new List<DhcpAzIpRangeScope>();
            IAzScopes scopes = GetAzApplication().Scopes;
            String strSubnetScope = azSubnetScope.GenerateAzScopeName();
            Regex filter = new Regex(@"^.*\s-\sIPR\s(\d.*):(\d.*)$");
            foreach (IAzScope scope in scopes)
            {
                try
                {
                    if (scope.Name.Contains(strSubnetScope + " - IPR"))
                    {
                        DhcpAzIpRangeScope ipr = new DhcpAzIpRangeScope();
                        ipr.Server = azSubnetScope.Server;
                        ipr.Subnet = azSubnetScope.Subnet;

                        Match m = filter.Match(scope.Name);
                        ipr.StartIp = new DhcpIpAddress(m.Groups[1].Value);
                        ipr.EndIp = new DhcpIpAddress(m.Groups[2].Value.Trim());
                        ranges.Add(ipr);
                    }
                }
                finally { FreeCom(scope, false); }
            }
            return ranges.ToArray();
        }

        private static object _Lock = new object();
        //private static void UpdateAzCache()
        //{
        //    lock (_Lock)
        //    {
        //
((AzAuthorizationStoreClass)HttpContext.Current.Application["AZ_STORE_OBJ"]).UpdateCac
he(null);
        //    }
        //}

        public static DhcpAzAccount[] GetPermissions(DhcpAzScope scope)
        {
            IAzScope azScope = null;
```

```
            List<DhcpAzAccount> accounts = new List<DhcpAzAccount>();
            try
            {

                //get scope based on name, null if does not exist
                azScope = DhcpSecurityOps.GetAzScope(scope.GenerateAzScopeName());

                //if scope doesn't exist, create it
                if (azScope == null)
                    return null;

                foreach (IAzRole azRole in azScope.Roles)
                {
                    if (azRole.Name == "Limited")
                        continue;

                    try
                    {
                        foreach (String accountName in ((Object[])azRole.MembersName))
                            accounts.Add(new DhcpAzAccount(accountName,
(DhcpAzRole)Enum.Parse(typeof(DhcpAzRole), azRole.Name, true)));
                    }
                    finally { FreeCom(azRole, false); }
                }

            }
            finally
            {
                FreeCom(azScope, false);
            }

            if (accounts.Count == 0)
                return null;
            return accounts.ToArray();
        }

        public static void AddPermission(DhcpAzScope scope, DhcpAzRole role, String
account)
        {

            //make limited role at subnet level for a new permission at ip level
            if (scope.ScopeType == DhcpAzScopeType.IpRange)
            {
                DhcpAzIpRangeScope azIpr = (DhcpAzIpRangeScope)scope;
                DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
                azSubnet.Server = azIpr.Server;
                azSubnet.Subnet = azIpr.Subnet;
                AddPermission(azSubnet, "Limited", account);
            }

            String strRole = Enum.GetName(typeof(DhcpAzRole), role);
            AddPermission(scope, strRole, account);
        }

        private static void AddPermission(DhcpAzScope scope, String role, String
account)
        {

            IAzScope azScope = null;
            try
            {
                lock (_Lock)
                {

                    //does scope exist, if not create (check to see if real?)
                    String scopeId = scope.GenerateAzScopeName();

                    //get scope based on name, null if does not exist
                    azScope = DhcpSecurityOps.GetAzScope(scopeId);
                    //if scope doesn't exist, create it
                    if (azScope == null)
                        azScope = DhcpSecurityOps.CreateCustomAzScope(scopeId);

                    IAzRole azRole = null;
                    try
                    {
                        //add account to role, if account already exists, COMException
is throw ErrorCode = 0x800700B7
                        azRole = azScope.OpenRole(role, null);
                        azRole.AddMemberName(account, null);
                        azRole.Submit(0, null);
                    }
                    catch (COMException ex)
                    {
                        //ERROR, User already exists in Role
                        if ((uint)ex.ErrorCode != 0x800700B7)
                            throw;

                    }
                    finally
                    {
                        FreeCom(azRole, false);
                    }

                }
            }
            finally
            {
                FreeCom(azScope, false);

            }
        }

        public static void RemovePermission(DhcpAzScope scope, DhcpAzRole role, String
account)
        {
            IAzScope azScope = null;
            Boolean forceGC = false;
            try
            {
                lock (_Lock)
                {

                    //Remove user or group from scope
                    String scopeId = scope.GenerateAzScopeName();
                    azScope = DhcpSecurityOps.GetAzScope(scopeId);

                    if (azScope == null)
                        return;

                    IAzRole azRole = null;
                    try
                    {
                        azRole = azScope.OpenRole(Enum.GetName(typeof(DhcpAzRole),
role), null);

                        azRole.DeleteMemberName(account, null);
                        azRole.Submit(0, null);
                    }
                    catch (COMException ex)
                    {
                        //ERROR, element not found
                        if ((uint)ex.ErrorCode != 0x80070490)
                            throw;

                    }
                    finally
                    {
                        FreeCom(azRole, false);
                    }

                    //if scope roles now empty, remove scope
                    if (areRolesEmpty(azScope))
                    {
                        forceGC = true;
                        DeleteCustomAzScope(scopeId);
                    }
                }

            }
            finally
            {
                FreeCom(azScope, forceGC);
            }

            //make limited role at subnet level for a new permission at ip level
            if (scope.ScopeType == DhcpAzScopeType.IpRange)
            {
                DhcpAzIpRangeScope azIpr = (DhcpAzIpRangeScope)scope;
                DhcpAzSubnetScope azSubnet = new DhcpAzSubnetScope();
                azSubnet.Server = azIpr.Server;
                azSubnet.Subnet = azIpr.Subnet;
                RemovePermission(azSubnet, "Limited", account);
            }
        }

        private static void RemovePermission(DhcpAzScope scope, String role, String
account)
        {
            IAzScope azScope = null;
            Boolean forceGC = false;
            try
            {
                lock (_Lock)
                {

                    //Remove user or group from scope
                    String scopeId = scope.GenerateAzScopeName();
                    azScope = DhcpSecurityOps.GetAzScope(scopeId);

                    if (azScope == null)
                        return;

                    IAzRole azRole = null;
                    try
                    {
                        azRole = azScope.OpenRole(role, null);
                        azRole.DeleteMemberName(account, null);
                        azRole.Submit(0, null);
                    }
                    catch (COMException ex)
                    {
                        //ERROR, element not found
                        if ((uint)ex.ErrorCode != 0x80070490)
                            throw;

                    }
                    finally
                    {
                        FreeCom(azRole, false);
                    }

                    //if scope roles now empty, remove scope
                    if (areRolesEmpty(azScope))
                    {
                        forceGC = true;
                        DeleteCustomAzScope(scopeId);
                    }
                }

            }
            finally
            {
                FreeCom(azScope, forceGC);
            }
        }

        public static void RemoveAzSubnetAzIpRanges(DhcpAzSubnetScope azSubnetScope)
        {
            DhcpAzIpRangeScope[] childrenScopes = EnumAzIpRanges(azSubnetScope);

            lock (_Lock)
            {
                //delete subnet az
                DeleteCustomAzScope(azSubnetScope.GenerateAzScopeName());

                //delete child ipr az
                foreach (DhcpAzIpRangeScope scope in childrenScopes)
                    DeleteCustomAzScope(scope.GenerateAzScopeName());
            }
        }

        public static void AddGlobalPermission(DhcpAzGlobalRole globalRole, String
account)
        {

            lock (_Lock)
            {

                IAzRole azRole = null;
                try
                {
                    //add account to role, if account already exists, COMException is
throw ErrorCode = 0x800700B7
                    //Add members to roles in scope
                    azRole =
GetAzApplication().OpenRole(Enum.GetName(typeof(DhcpAzGlobalRole), globalRole), null);
                    azRole.AddMemberName(account, null);
                    azRole.Submit(0, null);
                }
                catch (COMException ex)
                {
                    //ERROR, User already exists in Role
                    if ((uint)ex.ErrorCode != 0x800700B7)
                        throw;

                }
                finally
                {
                    FreeCom(azRole, false);
                }
            }

        }

        public static DhcpAzGlobalAccount[] GetGlobalPermissions()
        {

            List<DhcpAzGlobalAccount> accounts = new List<DhcpAzGlobalAccount>();

            IAzApplication2 azApp = GetAzApplication();
            foreach (IAzRole azRole in azApp.Roles)
            {
                try
                {
                    foreach (String accountName in ((Object[])azRole.MembersName))
```

```
                        accounts.Add(new DhcpAzGlobalAccount(accountName,
(DhcpAzGlobalRole)Enum.Parse(typeof(DhcpAzGlobalRole), azRole.Name, true)));
                }
                finally { FreeCom(azRole, false); }
            }

            if (accounts.Count == 0)
                return null;
            return accounts.ToArray();
        }

        public static void RemoveGlobalPermission(DhcpAzGlobalRole globalRole, String
account)
        {
            lock (_Lock)
            {
                IAzRole azRole = null;
                try
                {
                    azRole =
GetAzApplication().OpenRole(Enum.GetName(typeof(DhcpAzGlobalRole), globalRole), null);
                    azRole.DeleteMemberName(account, null);
                    azRole.Submit(0, null);
                }
                catch (COMException ex)
                {
                    //ERROR, element not found
                    if ((uint)ex.ErrorCode != 0x80070490)
                        throw;

                }
                finally
                {
                    FreeCom(azRole, false);
                }
            }
        }

        private static void DeleteCustomAzScope(String scope)
        {
            IAzApplication2 azApp = GetAzApplication();
            try
            {
                azApp.DeleteScope(scope, null);
                azApp.Submit(0, null);
            }
            catch { }
        }

        private static Boolean areRolesEmpty(IAzScope azScope)
        {
            Boolean flag = true;
            foreach (IAzRole azRole in azScope.Roles)
            {
                if (((Object[])azRole.MembersName).Length != 0)
                    flag = false;
                FreeCom(azRole, false);

                if (!flag)
                    break;
            }

            return flag;
        }

        private static void FreeCom(object o, bool forceGCwait)
        {
            try
            {
                Marshal.FinalReleaseComObject(o);
            }
            catch { }
            finally
            {
                o = null;
                if (forceGCwait)
                {
                    GC.Collect();
                    GC.WaitForPendingFinalizers();
                }
            }
        }

    }

    public enum DhcpAzOps
    {
        Traverse = 1,
        Read = 2,
        Create = 4,
        Update = 8,
        Delete = 16
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpSubnet.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;

using Unf.Dhcp.Smo.Interop;

namespace Unf.Dhcp.Smo
{
    public class DhcpSubnet
    {
        public DhcpIpAddress Address;
        public DhcpIpAddress Mask;
        public String Name;
        public String Comment;
        public DhcpSubnetState State;

        public DhcpSubnet() { this.State = DhcpSubnetState.Disabled; }

        public void Create(DhcpIpAddress server)
        {
            DHCP_SUBNET_INFO info = this.ConvertToNative();

            uint Response = NativeMethods.DhcpCreateSubnet(server.ToString(),
this.Address.GetUIntAddress(), ref info);

            if (Response != 0)
                throw new DhcpException(Response);
        }

        public void Delete(DhcpIpAddress server, bool force)
        {
            if (this.Address == null)
                throw new InvalidOperationException("Subnet invalid");
            uint Response = NativeMethods.DhcpDeleteSubnet(server.ToString(),
this.Address.GetUIntAddress(), (force ? DHCP_FORCE_FLAG.DhcpFullForce :
DHCP_FORCE_FLAG.DhcpNoForce));

            if (Response != 0)
                throw new DhcpException(Response);
        }

        public void Get(DhcpIpAddress server)
        {
            if (this.Address == null)
                throw new InvalidOperationException("Subnet invalid");

            IntPtr iPtr = IntPtr.Zero;
            UInt32 Response = 0;
            DHCP_SUBNET_INFO info = new DHCP_SUBNET_INFO();
            try
            {
                Response = NativeMethods.DhcpGetSubnetInfo(server.ToString(),
this.Address.GetUIntAddress(), out iPtr);

                info = (DHCP_SUBNET_INFO)Marshal.PtrToStructure(iPtr,
typeof(DHCP_SUBNET_INFO));
            }
            finally
            {
                if (iPtr != IntPtr.Zero)
                    NativeMethods.DhcpRpcFreeMemory(iPtr);
            }

            this.Address = new DhcpIpAddress(info.SubnetAddress);
            this.Mask = new DhcpIpAddress(info.SubnetMask);
            this.Name = info.SubnetName;
            this.Comment = info.SubnetComment;
            this.State = (DhcpSubnetState)info.SubnetState;

            if (Response != 0)
                throw new DhcpException(Response);

        }

        public void Update(DhcpIpAddress server)
        {
            DHCP_SUBNET_INFO info = this.ConvertToNative();

            uint Response = NativeMethods.DhcpSetSubnetInfo(server.ToString(),
this.Address.GetUIntAddress(), ref info);

            if (Response != 0)
                throw new DhcpException(Response);
        }

        public static List<DhcpSubnet> EnumAll(DhcpIpAddress server)
        {
            UInt32 Response = 0, ResumeHandle = 0;
            UInt32 nRead = 0, nTotal = 0;
            IntPtr retPtr;
            List<DhcpSubnet> subnets = new List<DhcpSubnet>();
            DhcpSubnet current;

            for( ; ; )
            {
                Response = NativeMethods.DhcpEnumSubnets(server.ToString(), ref
ResumeHandle, 1024, out retPtr,
                    out nRead, out nTotal);

                //ERROR_NO_MORE_ITEMS
                if (Response == 259)
                    break;
                if (Response != 0)
                    throw new DhcpException(Response);

                //work
                DHCP_IP_ARRAY NativeIps =
(DHCP_IP_ARRAY)Marshal.PtrToStructure(retPtr, typeof(DHCP_IP_ARRAY));

                for (int i = 0; i < NativeIps.NumElements; ++i)
                {
                    current = new DhcpSubnet();
                    current.Address = new
DhcpIpAddress((UInt32)Marshal.ReadInt32(NativeIps.Elements,
i*Marshal.SizeOf(typeof(UInt32))));
                    current.Get(server);
                    subnets.Add(current);
                }
            }

            //free on last successful call
            if (Response == 0)
                NativeMethods.DhcpRpcFreeMemory(retPtr);

            return subnets;
        }

        private DHCP_SUBNET_INFO ConvertToNative()
        {
            if (this.Address == null || this.Mask == null ||
String.IsNullOrEmpty(this.Name))
                throw new InvalidOperationException("Subnet invalid");

            DHCP_SUBNET_INFO nativeSubnet = new DHCP_SUBNET_INFO();
            nativeSubnet.SubnetAddress = this.Address.GetUIntAddress();
            nativeSubnet.SubnetMask = this.Mask.GetUIntAddress();
            nativeSubnet.SubnetName = this.Name;
            nativeSubnet.SubnetComment = this.Comment;
            nativeSubnet.SubnetState = (DHCP_SUBNET_STATE)this.State;
            return nativeSubnet;
        }

        private static DhcpSubnet ConvertFromNative(DHCP_SUBNET_INFO info)
        {
            DhcpSubnet subnet = new DhcpSubnet();
            subnet.Address = new DhcpIpAddress(info.SubnetAddress);
            subnet.Mask = new DhcpIpAddress(info.SubnetMask);
            subnet.Name = info.SubnetName;
            subnet.Comment = info.SubnetComment;
            subnet.State = (DhcpSubnetState)info.SubnetState;
            return subnet;
        }
    }

    public enum DhcpSubnetState
    {
        Enabled,
        Disabled,
        EnabledSwitched,
        DisabledSwitched,
        InvalidState
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: DhcpSubnetElement.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;

using Unf.Dhcp.Smo.Interop;

namespace Unf.Dhcp.Smo
{
    public class DhcpSubnetElement
    {
        public DhcpIpAddress SubnetAddress;
        public DhcpSubnetElementData Data;

        public DhcpSubnetElement() { }

        public void Add(DhcpIpAddress server)
        {
            if(this.SubnetAddress == null || Data == null)
                throw new InvalidOperationException("Invalid SubnetElement");

            uint Response = 0;
            using (MemManager Mem = new MemManager())
            {
                DHCP_SUBNET_ELEMENT_DATA_V5 element = this.Data.ConvertToNative(Mem);

                Response = NativeMethods.DhcpAddSubnetElementV5(server.ToString(),
this.SubnetAddress.GetUIntAddress(), ref element);
            }

            if (Response != 0)
                throw new DhcpException(Response);
        }

        public void Remove(DhcpIpAddress server, bool force)
        {
            if (this.SubnetAddress == null || Data == null)
                throw new InvalidOperationException("Invalid SubnetElement");

            uint Response = 0;
            using (MemManager Mem = new MemManager())
            {
                DHCP_SUBNET_ELEMENT_DATA_V5 element = this.Data.ConvertToNative(Mem);

                Response = NativeMethods.DhcpRemoveSubnetElementV5(server.ToString(),
this.SubnetAddress.GetUIntAddress(), ref element, (force ?
DHCP_FORCE_FLAG.DhcpFullForce : DHCP_FORCE_FLAG.DhcpNoForce));
            }

            if (Response != 0)
                throw new DhcpException(Response);

        }

        public DhcpSubnetElementData[] EnumAll(DhcpIpAddress server)
        {
            if (this.SubnetAddress == null || Data == null)
                throw new InvalidOperationException("Invalid SubnetElement");

            DHCP_SUBNET_ELEMENT_INFO_ARRAY_V5 NativeArray;
            List<DhcpSubnetElementData> elements = new List<DhcpSubnetElementData>();
            IntPtr retPtr, iPtr;
            UInt32 rHandle = 0, OptionsRead = 0, OptionsTotal = 0, Response = 0;

            for ( ; ; )
            {
                Response = NativeMethods.DhcpEnumSubnetElementsV5(server.ToString(),
this.SubnetAddress.GetUIntAddress(),
                    (DHCP_SUBNET_ELEMENT_TYPE_V5)this.Data.Type, ref rHandle,
UInt32.MaxValue, out retPtr, out OptionsRead, out OptionsTotal);

                //ERROR_NO_MORE_ITEMS
                if (Response == 259)
                    break;

                //ERROR_MORE_DATA = 234
                if (Response != 0 && Response != 234 || retPtr == IntPtr.Zero)
                {
                    if (retPtr != IntPtr.Zero)
                        NativeMethods.DhcpRpcFreeMemory(retPtr);
                    throw new DhcpException(Response);
                }

                //work
                NativeArray =
(DHCP_SUBNET_ELEMENT_INFO_ARRAY_V5)Marshal.PtrToStructure(retPtr,
typeof(DHCP_SUBNET_ELEMENT_INFO_ARRAY_V5));

                for (int i = 0; i < NativeArray.NumElements; ++i)
                {
                    iPtr = (IntPtr)(NativeArray.Elements.ToInt32() + (i *
Marshal.SizeOf(typeof(DHCP_SUBNET_ELEMENT_DATA_V5))));
                    DHCP_SUBNET_ELEMENT_DATA_V5 element =
(DHCP_SUBNET_ELEMENT_DATA_V5)Marshal.PtrToStructure(iPtr,
typeof(DHCP_SUBNET_ELEMENT_DATA_V5));
                    elements.Add(DhcpSubnetElementData.CovertFromNative(element));
                }

                //free on last successful call
                if (Response == 0)
                    NativeMethods.DhcpRpcFreeMemory(retPtr);
            }
            return elements.ToArray();
        }
    }

    #region SubnetElementData...
    public abstract class DhcpSubnetElementData
    {
        public DhcpSubnetElementDataType Type;
        public DhcpSubnetElementData() { }
        public DhcpSubnetElementData(DhcpSubnetElementDataType type) { this.Type =
type; }

        internal abstract DHCP_SUBNET_ELEMENT_DATA_V5 ConvertToNative(MemManager Mem);

        internal static DhcpSubnetElementData
CovertFromNative(DHCP_SUBNET_ELEMENT_DATA_V5 nativeElement)
        {
            switch (nativeElement.ElementType)
            {
                case DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpReservedIps:
                    return new DhcpIpReservation(nativeElement);

                case DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpIpRanges:
                case DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpExcludedIpRanges:
                case DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpIpRangesDhcpOnly:
                    return new DhcpIpRange(nativeElement);

                case DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpIpRangesBootpOnly:
                case DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpIpRangesDhcpBootp:
                    return new BootpIpRange(nativeElement);

                default:
                    throw new NotImplementedException("SubnetElementData Type: " +
nativeElement.ElementType);
            }
        }

    }

    public class DhcpIpRange : DhcpSubnetElementData
    {
        public DhcpIpAddress StartAddress;
        public DhcpIpAddress EndAddress;
        public DhcpIpRange() { }
        public DhcpIpRange(DhcpSubnetElementDataType type) : base(type) { }

        internal DhcpIpRange(DHCP_SUBNET_ELEMENT_DATA_V5 nativeElement)
        {

            this.Type = (DhcpSubnetElementDataType)nativeElement.ElementType;

            DHCP_IP_RANGE ipr =
(DHCP_IP_RANGE)Marshal.PtrToStructure(nativeElement.Data, typeof(DHCP_IP_RANGE));
            this.StartAddress = new DhcpIpAddress(ipr.StartAddress);
            this.EndAddress = new DhcpIpAddress(ipr.EndAddress);
        }

        internal override DHCP_SUBNET_ELEMENT_DATA_V5 ConvertToNative(MemManager Mem)
        {
            if (this.StartAddress == null)
                throw new InvalidOperationException("DhcpIpRange.StartAddress is
null");

            if (this.EndAddress == null)
                throw new InvalidOperationException("DhcpIpRange.EndAddress is null");

            DHCP_IP_RANGE dipr = new DHCP_IP_RANGE();
            dipr.StartAddress = this.StartAddress.GetUIntAddress();
            dipr.EndAddress = this.EndAddress.GetUIntAddress();

            DHCP_SUBNET_ELEMENT_DATA_V5 element = new DHCP_SUBNET_ELEMENT_DATA_V5();
            switch (this.Type)
            {
                case DhcpSubnetElementDataType.DhcpIpRanges:
                    element.ElementType = DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpIpRanges;
                    break;
                case DhcpSubnetElementDataType.DhcpExcludedIpRanges:
                    element.ElementType =
DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpExcludedIpRanges;
                    break;
                case DhcpSubnetElementDataType.DhcpIpRangesDhcpOnly:
                    element.ElementType =
DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpIpRangesDhcpOnly;
                    break;
                default:
                    throw new DhcpException(1629);
            }
            element.Data = Mem.AllocStruct(dipr);
            return element;

        }

    }

    public class BootpIpRange : DhcpIpRange
    {
        public UInt32 BootpAllocated;
        public UInt32 MaxBootpAllowed;
        public BootpIpRange() { this.MaxBootpAllowed = UInt32.MaxValue; }
        public BootpIpRange(DhcpSubnetElementDataType type) : base(type) {
this.MaxBootpAllowed = UInt32.MaxValue; }
```

```
        internal BootpIpRange(DHCP_SUBNET_ELEMENT_DATA_V5 nativeElement)
        {
            this.Type = (DhcpSubnetElementDataType)nativeElement.ElementType;

            DHCP_BOOTP_IP_RANGE ipr =
(DHCP_BOOTP_IP_RANGE)Marshal.PtrToStructure(nativeElement.Data,
typeof(DHCP_BOOTP_IP_RANGE));
            this.StartAddress = new DhcpIpAddress(ipr.StartAddress);
            this.EndAddress = new DhcpIpAddress(ipr.EndAddress);
            this.BootpAllocated = ipr.BootpAllocated;
            this.MaxBootpAllowed = ipr.MaxBootpAllowed;
        }

        internal override DHCP_SUBNET_ELEMENT_DATA_V5 ConvertToNative(MemManager Mem)
        {
            if (this.StartAddress == null)
                throw new InvalidOperationException("BootpIpRange.StartAddress is
null");

            if (this.EndAddress == null)
                throw new InvalidOperationException("BootpIpRange.EndAddress is
null");

            DHCP_BOOTP_IP_RANGE bipr = new DHCP_BOOTP_IP_RANGE();
            bipr.StartAddress = this.StartAddress.GetUIntAddress();
            bipr.EndAddress = this.EndAddress.GetUIntAddress();
            bipr.MaxBootpAllowed = this.MaxBootpAllowed;
            bipr.BootpAllocated = this.BootpAllocated;

            DHCP_SUBNET_ELEMENT_DATA_V5 element = new DHCP_SUBNET_ELEMENT_DATA_V5();
            switch (this.Type)
            {
                case DhcpSubnetElementDataType.DhcpIpRangesBootpOnly:
                    element.ElementType =
DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpIpRangesBootpOnly;
                    break;
                case DhcpSubnetElementDataType.DhcpIpRangesDhcpBootp:
                    element.ElementType =
DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpIpRangesDhcpBootp;
                    break;
                default:
                    throw new DhcpException(1629);
            }
            element.Data = Mem.AllocStruct(bipr);
            return element;
        }

    }

    public class DhcpIpReservation : DhcpSubnetElementData
    {
        public DhcpIpAddress ReservedIp;
        public DhcpMacAddress ReservedMac;
        public DhcpSubnetClientType bAllowedClientTypes;
        public DhcpIpReservation() { this.Type =
DhcpSubnetElementDataType.DhcpReservedIps; }

        internal DhcpIpReservation(DHCP_SUBNET_ELEMENT_DATA_V5 nativeElement)
        {
            this.Type = (DhcpSubnetElementDataType)nativeElement.ElementType;

            DHCP_IP_RESERVATION_V4 rez =
(DHCP_IP_RESERVATION_V4)Marshal.PtrToStructure(nativeElement.Data,
typeof(DHCP_IP_RESERVATION_V4));
            this.ReservedIp = new DhcpIpAddress(rez.ReservedIpAddress);
            this.bAllowedClientTypes = (DhcpSubnetClientType)rez.bAllowedClientTypes;
            DHCP_CLIENT_UID mac = 
(DHCP_CLIENT_UID)Marshal.PtrToStructure(rez.ReservedForClient, 
typeof(DHCP_CLIENT_UID));
            if (mac.DataLength < 5)
                return;
            byte[] bytes = new byte[(int)mac.DataLength - 5];
            Marshal.Copy((IntPtr)(mac.Data.ToInt32() + 5), bytes, 0, 
(int)mac.DataLength - 5);
            this.ReservedMac = new DhcpMacAddress(bytes);
        }

        internal override DHCP_SUBNET_ELEMENT_DATA_V5 ConvertToNative(MemManager Mem)
        {
            if (this.ReservedIp == null)
                throw new InvalidOperationException("IpReservation.ReservedIp is 
null");

            if (this.ReservedMac == null)
                throw new InvalidOperationException("IpReservation.ReservedMac is 
null");

            DHCP_IP_RESERVATION_V4 rip = new DHCP_IP_RESERVATION_V4();
            rip.ReservedIpAddress = this.ReservedIp.GetUIntAddress();
            rip.bAllowedClientTypes = (byte)this.bAllowedClientTypes;

            //extra work for mac address
            Byte[] mac = this.ReservedMac.GetByteArray();
            DHCP_CLIENT_UID uid = new DHCP_CLIENT_UID();
            uid.DataLength = (uint)mac.Length;
            uid.Data = Mem.AllocByteArray(this.ReservedMac.GetByteArray());
            rip.ReservedForClient = Mem.AllocStruct(uid);

            DHCP_SUBNET_ELEMENT_DATA_V5 element = new DHCP_SUBNET_ELEMENT_DATA_V5();
            switch (this.Type)
            {
                case DhcpSubnetElementDataType.DhcpReservedIps:
                    element.ElementType = DHCP_SUBNET_ELEMENT_TYPE_V5.DhcpReservedIps;
                    break;
                default:
                    throw new DhcpException(1629);
            }
            element.Data = Mem.AllocStruct(rip);
            return element;
        }
    }
    #endregion

    #region Enums...
    public enum DhcpSubnetElementDataType
    {
        DhcpIpRanges = 0,
        //DhcpSecondaryHosts = 1,
        DhcpReservedIps = 2,
        DhcpExcludedIpRanges = 3,
        //enum 4 is missing from documentation...
        DhcpIpRangesDhcpOnly = 5,
        DhcpIpRangesDhcpBootp = 6,
        DhcpIpRangesBootpOnly = 7
    }

    public enum DhcpSubnetClientType : byte
    {
        Dhcp = 1,
        Bootp = 2,
        Both = 3
    }
    #endregion
}
/*
```

```
 * DHCP Server Management Objects
 *
 *  File: Global.asax.cs
 *  Namespace: Unf.Dhcp.Smo.WebServices
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;
using Microsoft.Interop.Security.AzRoles;

namespace Unf.Dhcp.Smo.WebServices
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            Application.Lock();
            DhcpSecurityOps.Init();
            Application.UnLock();
        }
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: MemManager.cs
 *  Namespace: Unf.Dhcp.Smo
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;


namespace Unf.Dhcp.Smo.Interop
{
    internal class MemManager : IDisposable
    {
        public IntPtr AllocStruct(Object structure)
        {
            if (isDisposed)
                throw new ObjectDisposedException(this.ToString());

            IntPtr iptr = Marshal.AllocHGlobal(Marshal.SizeOf(structure));
            Marshal.StructureToPtr(structure, iptr, false);
            allocations.Add(iptr);
            return iptr;
        }

        public IntPtr AllocByteArray(Byte[] bytes)
        {
            if (isDisposed)
                throw new ObjectDisposedException(this.ToString());

            IntPtr iptr = Marshal.AllocHGlobal(bytes.Length);
            Marshal.Copy(bytes, 0, iptr, bytes.Length);
            allocations.Add(iptr);
            return iptr;
        }
        public IntPtr AllocArray(Type type, uint arraysize)
        {
            if (isDisposed)
                throw new ObjectDisposedException(this.ToString());

            IntPtr iptr = Marshal.AllocHGlobal((int)arraysize * Marshal.SizeOf(type));
            allocations.Add(iptr);
            return iptr;
        }

        public IntPtr AllocString(String str)
        {
            if (isDisposed)
                throw new ObjectDisposedException(this.ToString());

            IntPtr iptr = Marshal.StringToHGlobalAuto(str);
            allocations.Add(iptr);
            return iptr;
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!isDisposed)
            {
                //clean managed
                if (disposing) { }

                //clean unmanaged
                //Console.WriteLine("Disposing ("+ allocations.Count +")...");
                foreach (IntPtr i in this.allocations)
                    Marshal.FreeHGlobal(i);
                allocations.Clear();

            }
            isDisposed = true;
        }

        ~MemManager() { Dispose(false); }

        protected bool isDisposed = false;
        protected List<IntPtr> allocations = new List<IntPtr>();
    }
}
/*
 * DHCP Server Management Objects
 *
 *  File: NativeMethods.cs
 *  Namespace: Unf.Dhcp.Smo.Interop
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;

using System.Runtime.InteropServices;
using System.Net;
using System.Net.NetworkInformation;

namespace Unf.Dhcp.Smo.Interop
{
```

```
internal static class NativeMethods
{

    #region DhcpOptionFunctions...
    [DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
    internal static extern UInt32 DhcpEnumOptionsV5(
        string ServerIpAddress,
        UInt32 Flags,
        string ClassName,
        string VendorName,
        ref UInt32 ResumeHandle,
        UInt32 PreferredMaximum,
        out IntPtr Options,
        out UInt32 OptionsRead,
        out UInt32 OptionsTotal);

    [DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
    internal static extern UInt32 DhcpGetOptionInfoV5(
        string ServerIpAddress,
        UInt32 Flags,
        UInt32 OptionID,
        string ClassName,
        string VendorName,
        out IntPtr OptionInfo);
    #endregion

    #region DhcpOptionValueFunctions...

    [DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
    internal static extern UInt32 DhcpEnumOptionValuesV5(
        string ServerIpAddress,
        UInt32 Flags,
        string ClassName,
        string VendorName,
        ref DHCP_OPTION_SCOPE_INFO ScopeInfo,
        ref UInt32 ResumeHandle,
        UInt32 PreferredMaximum,
        out IntPtr OptionValues,
        out UInt32 OptionsRead,
        out UInt32 OptionsTotal);

    [DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
    internal static extern UInt32 DhcpRemoveOptionValueV5(
        string ServerIpAddress,
        UInt32 Flags,
        UInt32 OptionID,
        string ClassName,
        string VendorName,
        ref DHCP_OPTION_SCOPE_INFO ScopeInfo);

    [DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
    internal static extern UInt32 DhcpSetOptionValueV5(
        string ServerIpAddress,
        UInt32 Flags,
        UInt32 OptionID,
        string ClassName,
        string VendorName,
        ref DHCP_OPTION_SCOPE_INFO ScopeInfo,
        ref DHCP_OPTION_DATA OptionValue);

    [DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
    internal static extern UInt32 DhcpGetOptionValueV5(
        string ServerIpAddress,
        UInt32 Flags,
        UInt32 OptionID,
        string ClassName,
        string VendorName,
        ref DHCP_OPTION_SCOPE_INFO ScopeInfo,
        out IntPtr OptionValue);
```

```
#endregion

#region DhcpSubnetElementFunctions...
[DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
internal static extern UInt32 DhcpRemoveSubnetElementV5(
    string ServerIpAddress,
    UInt32 SubnetAddress,
    ref DHCP_SUBNET_ELEMENT_DATA_V5 RemoveElementInfo,
    DHCP_FORCE_FLAG ForceFlag);

[DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
internal static extern UInt32 DhcpAddSubnetElementV5(
    string ServerIpAddress,
    UInt32 SubnetAddress,
    ref DHCP_SUBNET_ELEMENT_DATA_V5 AddElementInfo);

[DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
internal static extern UInt32 DhcpEnumSubnetElementsV5(
    string ServerIpAddress,
    UInt32 SubnetAddress,
    DHCP_SUBNET_ELEMENT_TYPE_V5 EnumElementType,
    ref UInt32 ResumeHandle,
    UInt32 PreferredMaximum,
    out IntPtr EnumElementInfo,
    out UInt32 ElementsRead,
    out UInt32 ElementsTotal);
#endregion

#region DhcpClientFunctions...
[DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
internal static extern UInt32 DhcpDeleteClientInfo(
    string ServerIpAddress,
    ref DHCP_SEARCH_INFO SearchInfo);

[DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
internal static extern UInt32 DhcpGetClientInfo(
      String ServerIpAddress,
      ref DHCP_SEARCH_INFO SearchInfo,
      out IntPtr ClientInfo);

[DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
internal static extern UInt32 DhcpSetClientInfo(
    string ServerIpAddress,
    ref DHCP_CLIENT_INFO ClientInfo);

[DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
internal static extern UInt32 DhcpCreateClientInfo(
    string ServerIpAddress,
    ref DHCP_CLIENT_INFO ClientInfo);

[DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
internal static extern UInt32 DhcpEnumSubnetClients(
    string ServerIpAddress,
    UInt32 SubnetAddress,
    ref UInt32 ResumeHandle,
    UInt32 PreferredMaximum,
    out IntPtr ClientInfo,
    out UInt32 ElementsRead,
    out UInt32 ElementsTotal);

#endregion

#region DhcpSubnetFunctions...
[DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
internal static extern UInt32 DhcpSetSubnetInfo(
    string ServerIpAddress,
    UInt32 SubnetAddress,
    ref DHCP_SUBNET_INFO SubnetInfo);
```

```
        [DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
        internal static extern UInt32 DhcpDeleteSubnet(
            string ServerIpAddress,
            UInt32 SubnetAddress,
            DHCP_FORCE_FLAG ForceFlag);

        [DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
        internal static extern UInt32 DhcpCreateSubnet(
            string ServerIpAddress,
            UInt32 SubnetAddress,
            ref DHCP_SUBNET_INFO SubnetInfo);

        [DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
        internal static extern UInt32 DhcpEnumSubnets(
            string ServerIpAddress,
            ref UInt32 ResumeHandle,
            UInt32 PreferredMaximum,
            out IntPtr EnumInfo,
            out UInt32 ElementsRead,
            out UInt32 ElementsTotal);

        [DllImport("dhcpsapi.dll", CharSet = CharSet.Unicode)]
        internal static extern UInt32 DhcpGetSubnetInfo(
              string ServerIpAddress,
              UInt32 SubnetAddress,
              out IntPtr SubnetInfo);
        #endregion

        #region DhcpMisc...

        [DllImport("dhcpsapi.dll")]
        internal static extern void DhcpRpcFreeMemory(IntPtr BuffPtr);
        #endregion

    }

}
/*
 * DHCP Server Management Objects
 *
 *  File: NativeStructs.cs
 *  Namespace: Unf.Dhcp.Smo.Interop
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;
using System.Runtime.InteropServices;


namespace Unf.Dhcp.Smo.Interop
{
    #region DHCPArrays...

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_IP_ARRAY
    {
        public UInt32 NumElements;
        public IntPtr Elements;
    }

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_SUBNET_ELEMENT_INFO_ARRAY_V5
    {
        public UInt32 NumElements;
        public IntPtr Elements;
    }

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_OPTION_ARRAY
    {
        public UInt32 NumElements;
        public IntPtr Elements;
    }

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_OPTION_DATA
    {
        public UInt32 NumElements;
        public IntPtr Elements;
    }

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_CLIENT_INFO_ARRAY
    {
        public UInt32 NumElements;
        public IntPtr Elements;
    }

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_OPTION_VALUE_ARRAY
    {
        public UInt32 NumElements;
        public IntPtr Elements;
    }
    #endregion

    #region DHCPEnums...

    internal enum DHCP_SEARCH_INFO_TYPE
    {
        DhcpClientIpAddress,
        DhcpClientHardwareAddress,
        DhcpClientName
    }

    internal enum DHCP_OPTION_TYPE
    {
        DhcpUnaryElementTypeOption,
        DhcpArrayTypeOption
    }

    internal enum DHCP_SUBNET_STATE
    {
        DhcpSubnetEnabled,
        DhcpSubnetDisabled,
        DhcpSubnetEnabledSwitched,
        DhcpSubnetDisabledSwitched,
        DhcpSubnetInvalidState
    }


    internal enum DHCP_SUBNET_ELEMENT_TYPE_V5
    {
        DhcpIpRanges = 0,
        DhcpSecondaryHosts = 1,
        DhcpReservedIps = 2,
        DhcpExcludedIpRanges = 3,
        //value 4 is missing from documentation...
        //I figured this out because enumsubnetelements for enums below was cause
invalid parameter errors
        DhcpIpRangesDhcpOnly = 5,
        DhcpIpRangesDhcpBootp = 6,
        DhcpIpRangesBootpOnly = 7
```

```
}

internal enum DHCP_FORCE_FLAG
{
    DhcpFullForce,
    DhcpNoForce
}

internal enum DHCP_OPTION_SCOPE_TYPE
{
    DhcpDefaultOptions,
    DhcpGlobalOptions,
    DhcpSubnetOptions,
    DhcpReservedOptions,
    DhcpMScopeOptions
}

internal enum DHCP_OPTION_DATA_TYPE
{
    DhcpByteOption,
    DhcpWordOption,
    DhcpDWordOption,
    DhcpDWordDWordOption,
    DhcpIpAddressOption,
    DhcpStringDataOption,
    DhcpBinaryDataOption,
    DhcpEncapsulatedDataOption,
    DhcpIpv6AddressOption
}

#endregion

#region DhcpOptionDataElement...

[StructLayout(LayoutKind.Explicit, Size = 12)]
internal struct DHCP_OPTION_DATA_ELEMENT
{
    [FieldOffset(0)]
    public DHCP_OPTION_DATA_TYPE OptionType;
    [FieldOffset(4)]
    public byte ByteOption;
    [FieldOffset(4)]
    public UInt16 WordOption;
    [FieldOffset(4)]
    public UInt32 DWordOption;
    [FieldOffset(4)]
    public DWORD_DWORD DWordDWordOption;
    [FieldOffset(4)]
    public IntPtr StringOption;
    [FieldOffset(4)]
    public DHCP_BINARY_DATA BinDataOption;
}
#endregion

#region DhcpOptionScopeInfo...

[StructLayout(LayoutKind.Explicit, Size = 12)]
internal struct DHCP_OPTION_SCOPE_INFO
{
    [FieldOffset(0)]
    public DHCP_OPTION_SCOPE_TYPE ScopeType;
    [FieldOffset(4)]
    public UInt32 SubnetScopeInfo;
    [FieldOffset(4)]
    public DHCP_RESERVED_SCOPE ReservedScopeInfo;
    [FieldOffset(4)]
    public IntPtr GlobalScopeInfo;
}
#endregion
```

```
    #region DhcpSearchInfo...
    [StructLayout(LayoutKind.Explicit, Size = 12)]
    internal struct DHCP_SEARCH_INFO
    {
        [FieldOffset(0)]
        public DHCP_SEARCH_INFO_TYPE SearchType;
        [FieldOffset(4)]
        public DHCP_CLIENT_UID ClientHardwareAddress;
        [FieldOffset(4)]
        public UInt32 ClientIpAddress;
        [FieldOffset(4)]
        public IntPtr ClientName;
    }
    #endregion

    #region DhcpStructs...

    [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
    internal struct DHCP_SUBNET_INFO
    {
        public UInt32 SubnetAddress;
        public UInt32 SubnetMask;
        public String SubnetName;
        public String SubnetComment;
        public DHCP_HOST_INFO PrimaryHost;
        public DHCP_SUBNET_STATE SubnetState;
    }

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_IP_RESERVATION_V4
    {
        public UInt32 ReservedIpAddress;
        public IntPtr ReservedForClient; //DHCP_CLIENT_UID*
        public byte bAllowedClientTypes;
    }

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_BOOTP_IP_RANGE
    {
        public UInt32 StartAddress;
        public UInt32 EndAddress;
        public UInt32 BootpAllocated;
        public UInt32 MaxBootpAllowed;
        //A ULONG is a unsigned long in win32, a long is 32bits.  A ulong in c# is
64bits
    }

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_IP_RANGE
    {
        public UInt32 StartAddress;
        public UInt32 EndAddress;
    }

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_SUBNET_ELEMENT_DATA_V5
    {
        public DHCP_SUBNET_ELEMENT_TYPE_V5 ElementType;
        public IntPtr Data;
    }

    [StructLayout(LayoutKind.Sequential)]
    internal struct DHCP_RESERVED_SCOPE
    {
        public UInt32 ReservedIpAddress;
        public UInt32 ReservedIpSubnetAddress;
    }
```

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
internal struct DHCP_OPTION
{
    public UInt32 OptionID;
    public string OptionName;
    public string OptionComment;
    public DHCP_OPTION_DATA DefaultValue;
    public DHCP_OPTION_TYPE OptionType;
}

[StructLayout(LayoutKind.Sequential)]
internal struct DHCP_OPTION_VALUE
{
    public UInt32 OptionID;
    public DHCP_OPTION_DATA Value;
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
internal struct DHCP_HOST_INFO
{
    public UInt32 IpAddress;
    public string NetBiosName;
    public string HostName;
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
internal struct DHCP_CLIENT_INFO
{
    public UInt32 ClientIpAddress;
    public UInt32 SubnetMask;
    public DHCP_CLIENT_UID ClientHardwareAddress;
    public string ClientName;
    public string ClientComment;
    public DATE_TIME ClientLeaseExpires;
    public DHCP_HOST_INFO OwnerHost;
}

[StructLayout(LayoutKind.Sequential)]
internal struct DWORD_DWORD
{
    public UInt32 UpperWord1;
    public UInt32 LowerWord2;
}

[StructLayout(LayoutKind.Sequential)]
internal struct DHCP_CLIENT_UID
{
    public UInt32 DataLength;
    public IntPtr Data;
}

[StructLayout(LayoutKind.Sequential)]
internal struct DHCP_BINARY_DATA
{
    public UInt32 DataLength;
    public IntPtr Data;
}

[StructLayout(LayoutKind.Sequential)]
internal struct DATE_TIME
{
    public UInt32 dwLowDateTime;
    public UInt32 dwHighDateTime;

    public DATE_TIME(DateTime date)
    {
        if (date == DateTime.MaxValue)
        {
            this.dwLowDateTime = this.dwHighDateTime = UInt32.MaxValue;
```

```
                return;
            }
            if (date == DateTime.MinValue)
            {
                this.dwLowDateTime = this.dwHighDateTime = UInt32.MinValue;
                return;
            }

            this.dwLowDateTime = (UInt32)(date.ToFileTime() & 0xFFFFFFFF);
            this.dwHighDateTime = (UInt32)((date.ToFileTime() & 0x7FFFFFFF00000000) >>
32);
        }

        public DateTime Convert()
        {

            if (this.dwHighDateTime == 0 && this.dwLowDateTime == 0)
                return DateTime.MinValue;

            if (this.dwHighDateTime == int.MaxValue && this.dwLowDateTime ==
UInt32.MaxValue)
                return DateTime.MaxValue;

            return DateTime.FromFileTime((((long)this.dwHighDateTime) << 32) |
(UInt32)this.dwLowDateTime);
        }

        public override string ToString()
        {
            return this.Convert().ToString();
        }
    }
    #endregion


    #region DhcpsapiError

    internal enum DhcpsapiErrorType : uint
    {
        [Description("The DHCP server registry initialization parameters are
incorrect.")]
        ERROR_DHCP_REGISTRY_INIT_FAILED = 20000,
        [Description("The DHCP server was unable to open the database of DHCP
clients.")]
        ERROR_DHCP_DATABASE_INIT_FAILED = 20001,
        [Description("The DHCP server was unable to start as a Remote Procedure Call
(RPC) server.")]
        ERROR_DHCP_RPC_INIT_FAILED = 20002,
        [Description("The DHCP server was unable to establish a socket connection.")]
        ERROR_DHCP_NETWORK_INIT_FAILED = 20003,
        [Description("The specified subnet already exists on the DHCP server.")]
        ERROR_DHCP_SUBNET_EXISTS = 20004,
        [Description("The specified subnet does not exist on the DHCP server.")]
        ERROR_DHCP_SUBNET_NOT_PRESENT = 20005,
        [Description("The primary host information for the specified subnet was not
found on the DHCP server.")]
        ERROR_DHCP_PRIMARY_NOT_FOUND = 20006,
        [Description("The specified DHCP element has been used by a client and cannot
be removed.")]
        ERROR_DHCP_ELEMENT_CANT_REMOVE = 20007,
        [Description("The specified option already exists on the DHCP server.")]
        ERROR_DHCP_OPTION_EXISTS = 20009,
        [Description("The specified option does not exist on the DHCP server.")]
        ERROR_DHCP_OPTION_NOT_PRESENT = 20010,
        [Description("The specified IP address is not available.")]
        ERROR_DHCP_ADDRESS_NOT_AVAILABLE = 20011,
        [Description("The specified IP address range has all of its member addresses
leased.")]
        ERROR_DHCP_RANGE_FULL = 20012,
        [Description("An error occurred while accessing the DHCP JET database. For
more information about this error, please look at the DHCP server event log.")]
        ERROR_DHCP_JET_ERROR = 20013,
        [Description("The specified client already exists in the database.")]
        ERROR_DHCP_CLIENT_EXISTS = 20014,
        [Description("The DHCP server received an invalid message.")]
        ERROR_DHCP_INVALID_DHCP_MESSAGE = 20015,
        [Description("The DHCP server received an invalid message from the client.")]
        ERROR_DHCP_INVALID_DHCP_CLIENT = 20016,
        [Description("The DHCP server is currently paused.")]
        ERROR_DHCP_SERVICE_PAUSED = 20017,
        [Description("The specified DHCP client is not a reserved client.")]
        ERROR_DHCP_NOT_RESERVED_CLIENT = 20018,
        [Description("The specified DHCP client is a reserved client.")]
        ERROR_DHCP_RESERVED_CLIENT = 20019,
        [Description("The specified IP address range is too small.")]
        ERROR_DHCP_RANGE_TOO_SMALL = 20020,
        [Description("The specified IP address range is already defined on the DHCP
server.")]
        ERROR_DHCP_IPRANGE_EXISTS = 20021,
        [Description("The specified Reservation is currently taken by another
client.")]
        ERROR_DHCP_RESERVEDIP_EXISTS = 20022,
        [Description("The specified IP address range either overlaps with an existing
range or is invalid.")]
        ERROR_DHCP_INVALID_RANGE = 20023,
        [Description("The specified IP address range is an extension of an existing
range.")]
        ERROR_DHCP_RANGE_EXTENDED = 20024,
        [Description("The specified IP address range extension is too small. The
number of addresses in the extension must be a multiple of 32.")]
        ERROR_DHCP_RANGE_EXTENSION_TOO_SMALL = 20025,
        [Description("An attempt was made to extend the IP address range to a value
less than the specified backward extension. The number of addresses in the extension
must be a multiple of 32.")]
        ERROR_DHCP_WARNING_RANGE_EXTENDED_LESS = 20026,
        [Description("The DHCP database needs to be upgraded to a newer format. For
more information, refer to the DHCP server event log.")]
        ERROR_DHCP_JET_CONV_REQUIRED = 20027,
        [Description("The format of the bootstrap protocol file table is incorrect.")]
        ERROR_DHCP_SERVER_INVALID_BOOT_FILE_TABLE = 20028,
        [Description("A boot file name specified in the bootstrap protocol file table
is unrecognized or invalid.")]
        ERROR_DHCP_SERVER_UNKNOWN_BOOT_FILE_NAME = 20029,
        [Description("The specified superscope name is too long.")]
        ERROR_DHCP_SUPER_SCOPE_NAME_TOO_LONG = 20030,
        [Description("The specified IP address is already in use.")]
        ERROR_DHCP_IP_ADDRESS_IN_USE = 20032,
        [Description("The specified path to the DHCP audit log file is too long.")]
        ERROR_DHCP_LOG_FILE_PATH_TOO_LONG = 20033,
        [Description("The DHCP server received a request for a valid IP address not
administered by the server.")]
        ERROR_DHCP_UNSUPPORTED_CLIENT = 20034,
        [Description("The DHCP server failed to receive a notification when the
interface list changed, therefore some of the interfaces will not be enabled on the
server.")]
        ERROR_DHCP_SERVER_INTERFACE_NOTIFICATION_EVENT = 20035,
        [Description("The DHCP database needs to be upgraded to a newer format
(JET97). For more information, refer to the DHCP server event log.")]
        ERROR_DHCP_JET97_CONV_REQUIRED = 20036,
        [Description("The DHCP server cannot determine if it has the authority to run,
and is not servicing clients on the network. This rogue status may be due to network
problems or insufficient server resources.")]
        ERROR_DHCP_ROGUE_INIT_FAILED = 20037,
        [Description("The DHCP service is shutting down because another DHCP server is
active on the network.")]
        ERROR_DHCP_ROGUE_SAMSHUTDOWN = 20038,
        [Description("The DHCP server does not have the authority to run, and is not
servicing clients on the network.")]
```

```
        ERROR_DHCP_ROGUE_NOT_AUTHORIZED = 20039,
        [Description("The DHCP server is unable to contact the directory service for
this domain. The DHCP server will continue to attempt to contact the directory
service. During this time, no clients on the network will be serviced.")]
        ERROR_DHCP_ROGUE_DS_UNREACHABLE = 20040,
        [Description("The DHCP server's authorization information conflicts with that
of another DHCP server on the network.")]
        ERROR_DHCP_ROGUE_DS_CONFLICT = 20041,
        [Description("The DHCP server is ignoring a request from another DHCP server
because the second server is a member of a different directory service enterprise.")]
        ERROR_DHCP_ROGUE_NOT_OUR_ENTERPRISE = 20042,
        [Description("The DHCP server has detected a directory service environment on
the network. If there is a directory service on the network, the DHCP server can only
run if it is a part of the directory service. Since the server ostensibly belongs to a
workgroup, it is terminating.")]
        ERROR_DHCP_STANDALONE_IN_DS = 20043,
        [Description("The specified DHCP class name is unknown or invalid.")]
        ERROR_DHCP_CLASS_NOT_FOUND = 20044,
        [Description("The specified DHCP class name (or information) is already in
use.")]
        ERROR_DHCP_CLASS_ALREADY_EXISTS = 20045,
        [Description("The specified DHCP scope name is too long; the scope name must
not exceed 256 characters.")]
        ERROR_DHCP_SCOPE_NAME_TOO_LONG = 20046,
        [Description("The default scope is already configured on the server.")]
        ERROR_DHCP_DEFAULT_SCOPE_EXISTS = 20047,
        [Description("The Dynamic BOOTP attribute cannot be turned on or off.")]
        ERROR_DHCP_CANT_CHANGE_ATTRIBUTE = 20048,
        [Description("Conversion of a scope to a \"DHCP Only\" scope or to a \"BOOTP
Only\" scope is not allowed when the scope contains other DHCP and BOOTP clients.
Either the DHCP or BOOTP clients should be specifically deleted before converting the
scope to the other type.")]
        ERROR_DHCP_IPRANGE_CONV_ILLEGAL = 20049,
        [Description("The network has changed. Retry this operation after checking for
network changes. Network changes may be caused by interfaces that are new or invalid,
or by IP addresses that are new or invalid.")]
        ERROR_DHCP_NETWORK_CHANGED = 20050,
        [Description("The bindings to internal IP addresses cannot be modified.")]
        ERROR_DHCP_CANNOT_MODIFY_BINDINGS = 20051,
        [Description("The DHCP scope parameters are incorrect. Either the scope
already exists, or its properties are inconsistent with the subnet address and mask of
an existing scope.")]
        ERROR_DHCP_SUBNET_EXISTS_2 = 20052,
        [Description("The DHCP multicast scope parameters are incorrect. Either the
scope already exists, or its properties are inconsistent with the subnet address and
mask of an existing scope.")]
        ERROR_DHCP_MSCOPE_EXISTS = 20053,
        [Description("The multicast scope range must have at least 256 IP
addresses.")]
        ERROR_DHCP_MSCOPE_RANGE_TOO_SMALL = 20054,
        [Description("The DHCP server could not contact Active Directory.")]
        ERROR_DDS_NO_DS_AVAILABLE = 20070,
        [Description("The DHCP service root could not be found in Active Directory.")]
        ERROR_DDS_NO_DHCP_ROOT = 20071,
        [Description("An unexpected error occurred while accessing Active
Directory.")]
        ERROR_DDS_UNEXPECTED_ERROR = 20072,
        [Description("There were too many errors to proceed.")]
        ERROR_DDS_TOO_MANY_ERRORS = 20073,
        [Description("A DHCP service could not be found.")]
        ERROR_DDS_DHCP_SERVER_NOT_FOUND = 20074,
        [Description("The specified DHCP options are already present in Active
Directory.")]
        ERROR_DDS_OPTION_ALREADY_EXISTS = 20075,
        [Description("The specified DHCP options are not present in Active
Directory.")]
        ERROR_DDS_OPTION_DOES_NOT_EXIST = 20076,
        [Description("The specified DHCP classes are already present in Active
Directory.")]
```

```
        ERROR_DDS_CLASS_EXISTS = 20077,
        [Description("The specified DHCP classes are not present in Active
Directory.")]
        ERROR_DDS_CLASS_DOES_NOT_EXIST = 20078,
        [Description("The specified DHCP servers are already present in Active
Directory.")]
        ERROR_DDS_SERVER_ALREADY_EXISTS = 20079,
        [Description("The specified DHCP servers are not present in Active
Directory.")]
        ERROR_DDS_SERVER_DOES_NOT_EXIST = 20080,
        [Description("The specified DHCP server address does not correspond to the
identified DHCP server name.")]
        ERROR_DDS_SERVER_ADDRESS_MISMATCH = 20081,
        [Description("The specified subnets are already present in Active
Directory.")]
        ERROR_DDS_SUBNET_EXISTS = 20082,
        [Description("The specified subnet belongs to a different superscope.")]
        ERROR_DDS_SUBNET_HAS_DIFF_SUPER_SCOPE = 20083,
        [Description("The specified subnet is not present in Active Directory.")]
        ERROR_DDS_SUBNET_NOT_PRESENT = 20084,
        [Description("The specified reservation is not present in Active Directory.")]
        ERROR_DDS_RESERVATION_NOT_PRESENT = 20085,
        [Description("The specified reservation conflicts with another reservation
present in Active Directory.")]
        ERROR_DDS_RESERVATION_CONFLICT = 20086,
        [Description("The specified IP address range conflicts with another IP range
present in Active Directory.")]
        ERROR_DDS_POSSIBLE_RANGE_CONFLICT = 20087,
        [Description("The specified IP address range is not present in Active
Directory.")]
        ERROR_DDS_RANGE_DOES_NOT_EXIST = 20088
    }

    internal static class DhcpsapiError
    {
        public static String GetDesc(DhcpsapiErrorType e)
        {
            System.Reflection.FieldInfo EnumInfo = e.GetType().GetField(e.ToString());
            System.ComponentModel.DescriptionAttribute[] EnumAttributes =

(System.ComponentModel.DescriptionAttribute[])EnumInfo.GetCustomAttributes(typeof(Syst
em.ComponentModel.DescriptionAttribute), false);

            if (EnumAttributes.Length > 0) { return EnumAttributes[0].Description; }

            return "DHCPSAPI Error Unknown";
        }
    }

    #endregion

}
/*
 * DHCP Server Management Objects
 *
 *  File: ServerOperations.cs
 *  Namespace: Unf.Dhcp.Smo.Operations
 *
 * (c) Jason Rupard, School of Computing, University of North Florida
 *
*/
using System;
using System.Collections.Generic;
using System.Text;

using Unf.Dhcp.Smo;

namespace Unf.Dhcp.Smo.Operations
{
    public static class ServerOperations
    {
        #region Subnet Ops...
        public static List<DhcpSubnet> EnumSubnets(DhcpIpAddress server)
        {
            return DhcpSubnet.EnumAll(server);
        }

        public static void CreateSubnet(DhcpIpAddress server, DhcpSubnet network,
DhcpIpRange ipRange)
        {

            if (ipRange.Type == DhcpSubnetElementDataType.DhcpExcludedIpRanges ||
                ipRange.Type == DhcpSubnetElementDataType.DhcpReservedIps)
                throw new Exception("SubnetElementData incorrect type");

            //create subnet
            network.Create(server);

            //assign default range
            DhcpSubnetElement se = new DhcpSubnetElement();
            se.SubnetAddress = network.Address;
            se.Data = ipRange;
            se.Add(server);

            //zero out DNS option
            DhcpOptionValue ov = new DhcpOptionValue();
            ov.ClassType = DhcpOptionClassType.Dhcp;
            ov.OptionId = 81;
            DhcpOptionScope os = new DhcpOptionScope();
            os.type = DhcpOptionScopeType.Subnet;
            os.SubnetIp = network.Address;
            ov.OptionScope = os;
            ov.OptionData = new DhcpOptionDataDword(new UInt32[] { 0 });
            ov.Set(server);

        }

        public static void UpdateSubnet(DhcpIpAddress server, DhcpSubnet network)
        {
            //update subnet
            network.Update(server);
        }

        public static void UpdateSubnetRange(DhcpIpAddress server, DhcpIpAddress
network, DhcpIpRange ipRange)
        {
            if (ipRange.Type == DhcpSubnetElementDataType.DhcpExcludedIpRanges ||
                ipRange.Type == DhcpSubnetElementDataType.DhcpReservedIps)
                throw new Exception("SubnetElementData incorrect type");

            //Remove old ipRange
            DhcpSubnetElement se = new DhcpSubnetElement();
            se.SubnetAddress = network;
            se.Data = new
BootpIpRange(DhcpSubnetElementDataType.DhcpIpRangesDhcpBootp);
            DhcpSubnetElementData[] array = se.EnumAll(server);
            se.Data = array[0];
            se.Remove(server, true);

            try
            {
                //Add new ranage
                se.Data = ipRange;
                se.Add(server);
            }
            catch (Exception e)
            {
                //rollback
                se.Data = array[0];
                se.Add(server);
                throw e;
            }
        }

        public static void DeleteSubnet(DhcpIpAddress server, DhcpIpAddress network)
        {
            DhcpSubnet sub = new DhcpSubnet();
            sub.Address = network;
            sub.Delete(server, true);
        }

        public static void CreateSubnetExclusion(DhcpIpAddress server, DhcpIpAddress
network, DhcpIpRange ipExRange)
        {
            if(ipExRange.Type != DhcpSubnetElementDataType.DhcpExcludedIpRanges)
                throw new Exception("SubnetElementData incorrect type");

            //Add Exclusion Range
            DhcpSubnetElement se = new DhcpSubnetElement();
            se.SubnetAddress = network;
            se.Data = ipExRange;
            se.Add(server);
        }

        public static void DeleteSubnetExclusion(DhcpIpAddress server, DhcpIpAddress
network, DhcpIpRange ipExRange)
        {
            if (ipExRange.Type != DhcpSubnetElementDataType.DhcpExcludedIpRanges)
                throw new Exception("SubnetElementData incorrect type");

            //Add Exclusion Range
            DhcpSubnetElement se = new DhcpSubnetElement();
            se.SubnetAddress = network;
            se.Data = ipExRange;
            se.Remove(server, true);
        }

        public static List<DhcpClient> SubnetFindAllClients(DhcpIpAddress server,
DhcpIpAddress network, DhcpClientFilter terms)
        {
            return DhcpClient.EnumAll(server, network).FindAll(terms.Filter);
        }

        public static DhcpClient SubnetFindOneClient(DhcpIpAddress server,
DhcpIpAddress network, DhcpClientFilter terms)
        {
            return DhcpClient.EnumAll(server, network).Find(terms.Filter);
        }

        public static List<DhcpClient> ServerFindAllClients(DhcpIpAddress server,
DhcpClientFilter terms)
        {
            List<DhcpSubnet> subnets = DhcpSubnet.EnumAll(server);

            List<DhcpClient> clients = new List<DhcpClient>();
            foreach (DhcpSubnet net in subnets)
            {
                 clients.AddRange(DhcpClient.EnumAll(server,
net.Address).FindAll(terms.Filter));
            }
            return clients;
        }

        public static DhcpClient ServerFindOneClient(DhcpIpAddress server,
DhcpClientFilter terms)
        {
            List<DhcpSubnet> subnets = DhcpSubnet.EnumAll(server);
            DhcpClient client = null;
            foreach (DhcpSubnet net in subnets)
            {
                client = DhcpClient.EnumAll(server, net.Address).Find(terms.Filter);
                if(client != null)
                    return client;
            }
            return client;
        }

        public static DhcpIpRange[] EnumPools(DhcpIpAddress server, DhcpIpAddress
network)
        {
            List<DhcpIpRange> pools = new List<DhcpIpRange>(3);
            DhcpSubnetElement se = new DhcpSubnetElement();
            se.SubnetAddress = network;
            //use DhcpIpRangesDhcpBootp to get Main Range of any type.
            se.Data = new
DhcpIpRange(DhcpSubnetElementDataType.DhcpIpRangesDhcpBootp);
            DhcpSubnetElementData[] mainpool = se.EnumAll(server);
            pools.Add((DhcpIpRange)mainpool[0]);

            se.Data = new DhcpIpRange(DhcpSubnetElementDataType.DhcpExcludedIpRanges);
            DhcpSubnetElementData[] exPools = se.EnumAll(server);
            foreach (DhcpSubnetElementData pool in exPools)
                pools.Add((DhcpIpRange)pool);

            return pools.ToArray();
        }
        #endregion

        #region Lease Ops...
        public static List<DhcpClient> EnumLeases(DhcpIpAddress server, DhcpIpAddress
network)
        {
            return DhcpClient.EnumAll(server, network);
        }

        public static void CreateLease(DhcpIpAddress server, DhcpClient lease)
        {
            lease.Create(server);
        }

        public static void UpdateLease(DhcpIpAddress server, DhcpClient lease)
        {
            lease.Update(server);
        }

        public static void DeleteLease(DhcpIpAddress server, DhcpIpAddress leaseIp)
        {
            DhcpSearchInfo term = new DhcpSearchInfo();
            term.Type = DhcpSearchInfoType.IpAddress;
            term.IpAddress = leaseIp;
            DhcpClient.Delete(server, term);
        }

        #endregion

        #region Rez Ops...

        public static void CreateReservation(DhcpIpAddress server, DhcpIpAddress
network, DhcpReservation res)
        {
            //get subnet object for mask
            DhcpSubnet snet = new DhcpSubnet();
            snet.Address = network;
            snet.Get(server);
            //set internal res and lease objects
            DhcpIpReservation r = new DhcpIpReservation();
            DhcpClient c = new DhcpClient();
            r.ReservedIp = c.IpAddress = res.ReservedIp;
            r.ReservedMac = c.MacAddress = res.ReservedMac;
            r.bAllowedClientTypes = res.bAllowedClientTypes;
            c.SubnetMask = snet.Mask;
            c.Name = res.Name;
            c.Comment = res.Comment;

            //create element (reservation) to subnet
            DhcpSubnetElement se = new DhcpSubnetElement();
            se.SubnetAddress = network;
            se.Data = (DhcpSubnetElementData)r;
            se.Add(server);

            //update lease data
            c.Update(server);

            //zero out DNS option
            DhcpOptionValue ov = new DhcpOptionValue();
            ov.ClassType = DhcpOptionClassType.Dhcp;
            ov.OptionId = 81;
            DhcpOptionScope os = new DhcpOptionScope();
            os.type = DhcpOptionScopeType.Reservation;
            os.SubnetIp = network;
            os.ReservationIp = res.ReservedIp;
            ov.OptionScope = os;
            ov.OptionData = new DhcpOptionDataDword(new UInt32[] { 0 });
            ov.Set(server);

        }

        public static void UpdateReservation(DhcpIpAddress server, DhcpIpAddress
network, DhcpReservation res)
        {
            //get subnet object for mask
            DhcpSubnet snet = new DhcpSubnet();
            snet.Address = network;
            snet.Get(server);

            DhcpIpReservation r = new DhcpIpReservation();
            DhcpClient c = new DhcpClient();
            r.ReservedIp = c.IpAddress = res.ReservedIp;
            r.ReservedMac = c.MacAddress = res.ReservedMac;
            r.bAllowedClientTypes = res.bAllowedClientTypes;
            c.SubnetMask = snet.Mask;
            c.Name = res.Name;
            c.Comment = res.Comment;

            DhcpSubnetElement se = new DhcpSubnetElement();
            se.SubnetAddress = network;
            se.Data = (DhcpSubnetElementData)r;
            se.Remove(server, false);
            se.Add(server);
            c.Update(server);


        }

        public static void DeleteReservation(DhcpIpAddress server, DhcpIpAddress
network, DhcpIpAddress resIp)
        {
            DhcpIpReservation r = new DhcpIpReservation();
            r.ReservedIp = resIp;
            r.ReservedMac = new DhcpMacAddress();

            DhcpSubnetElement se = new DhcpSubnetElement();
            se.SubnetAddress = network;
            se.Data = (DhcpSubnetElementData)r;
            se.Remove(server, true);

        }

        public static List<DhcpReservation> EnumReservations(DhcpIpAddress server,
DhcpIpAddress network)
        {
            DhcpSubnetElement se = new DhcpSubnetElement();
            se.SubnetAddress = network;
            se.Data = new DhcpIpReservation();
            DhcpSubnetElementData[] resArray = se.EnumAll(server);
            List<DhcpReservation> ResList = new
List<DhcpReservation>(resArray.Length);

            DhcpClient c;
            DhcpSearchInfo term = new DhcpSearchInfo();
            term.Type = DhcpSearchInfoType.IpAddress;
            foreach (DhcpIpReservation res in resArray)
            {
                term.IpAddress = res.ReservedIp;
                c = DhcpClient.Get(server, term);

                DhcpReservation resinfo = new DhcpReservation();
                resinfo.Name = c.Name;
                resinfo.Comment = c.Comment;
                resinfo.ReservedIp = res.ReservedIp;
                resinfo.ReservedMac = res.ReservedMac;
                resinfo.bAllowedClientTypes = res.bAllowedClientTypes;

                ResList.Add(resinfo);
            }
            return ResList;
        }
        #endregion

        #region Option Ops...

        public static void SetOptionValue(DhcpIpAddress server, DhcpOptionValue value)
        {
            value.Set(server);
        }

        public static DhcpOptionValue GetOptionValue(DhcpIpAddress server, UInt32
optionId, DhcpOptionClassType classType, DhcpOptionScope scope)
        {
            DhcpOptionValue value = new DhcpOptionValue();
            value.OptionId = optionId;
            value.ClassType = classType;
            value.OptionScope = scope;
            value.Get(server);
            return value;
        }

        public static void RemoveOptionValue(DhcpIpAddress server, DhcpOptionValue
value)
        {
            value.Remove(server);
        }

        public static DhcpOptionValue[] EnumOptionValues(DhcpIpAddress server,
DhcpOptionClassType classType, DhcpOptionScope scope)
        {
            return DhcpOptionValue.EnumAll(server, classType, scope);
        }

        public static DhcpOption[] EnumOptions(DhcpIpAddress server,
DhcpOptionClassType classType)
        {
            return DhcpOption.EnumAll(server, classType);
        }
        #endregion
    }}
```

APPENDIX B

DHPC Web Services Installation

Overview
DHCP Web Services (DWS) contains two sets of ASP.NET 2.0 web services, DhcpOperations.asmx and DhcpSecurity.asmx. Both sets of web services implement SOAP messages based on the WS-I standard, Basic Profile 1.1. DhcpOperations provides a set of operations to manage a Microsoft DHCP server. While DhcpSecurity provides operations to manage the authorization layer within the DhcpOperations web services. DWS employs a trusted subsystem application model. The connecting client account to a web service must be authenticated (via IIS) and authorized (via DWS itself) to perform a DHCP operation. Once a client is authorized, a service account with access to the DHCP server will perform the operation on the client's behalf.

Terms

Term	Description
webserver	Hostname of IIS web server hosting DWS application
dhcpserver	Hostname of existing Windows 2003 DHCP server
serviceaccount	Local or Domain user account used as identity for the DWS application pool(*webserver*) and also authorized as a DHCP Administrator (*dhcpserver*) Examples: *domain*\user, *localhost*\user
domain	Active Directory Domain
localhost	In a non-domain install scenario, this is the server that hosts the DHCP _and_ the DWS application
dwsroot	Location of DHCP Web Services application Examples: C:\dws, C:\inetpub\wwwroot\dws

Systems Architecture
Web Server on DHCP Server (Single Server Deployment)
The IIS web server and DHCP server are located on the same machine. This scenario has its advantage when the machine is not deployed in an Active Directory domain. Local accounts can be used as *serviceaccount* and end user client authorizations.

Separate Web and DHCP Server
The IIS web server is deployed on a separate machine than the existing DHCP server. Domain accounts must be used as the *serviceaccount* and end user client authorizations in this scenario.

Requirements
Windows Server 2003 SP1 or greater
Microsoft .NET 2.0
IIS 6.0 with ASP.NET 2.0 enabled

Installation
Installation begins by verifying ASP.NET 2.0 is installed properly on *webserver*, then setting up *serviceaccount* rights and creating an application pool for DWS. Finally, choose a location to copy DWS application, *dwsroot*, and configure IIS settings for DWS. DWS will use basic authentication over SSL for user authentication and Authorization Manager (AzMan) for DHCP authorizations. Finally, an authorization will be set for the DWS global administrator role, this account can begin to use DWS and/or start assigning additional authorizations so other users can consume DWS.

Verify ASP.NET 2.0
On *webserver*

- Open IIS MMC, select ‘Web Service Extensions’
- If ‘ASP.NET v2.0.x’ missing or not in allowed state, perform step 3
- Register ASP.NET in IIS. At command prompt:

  ```
  %systemroot%\Microsoft.NET\Framework\v2.0.x\aspnet_regiis -i
  ```

Setup Service Account
On *webserver*

- Register *serviceaccount* for the ability to run a ASP.NET 2.0 application properly:

  ```
  %systemroot%\Microsoft.NET\Framework\v2.0.x\aspnet_regiis –ga
  domain\serviceaccount
  ```
- Add modify ACL to %systemroot%\temp directory for local computer group IIS_WPG

On *dhcpserver*

- Add *serviceaccount* to local computer group DHCP Administrators

Create Application Pool
On *webserver*

- Open IIS MMC, right client ‘Application Pools’ → New → Application Pool…
  - Name: DWSAppPool, OK
- Assign *serviceaccount* to app pool identity: Right client new pool → properties → Identity Tab
  - Select ‘Configurable’
  - User: *domain\serviceaccount*

Install and Configure DWS
On *webserver*

- Copy dws directory from DWS media to *dwsroot*
  - If *dwsroot* is located in website’s root directory (usually C:\Inetpub\wwwroot), skip to step X. Otherwise, create a IIS virtual directory to DWS location outlined in next steps
- Create Virtual Directory (IIS MMC)
  - Right click website → New → Virtual Directory…
    - Alias: dws
    - Path: *dwsroot*
    - Access Permissions: skip
- Configure IIS setting for DWS directory, right client DWS folder in website tree → properties
  - Virtual Directory Tab (Application Settings)
    - Select ‘Create’ button
    - Execute Permissions: Scripts and Executables
    - Application Pool: DWSAppPool
  - ASP.NET Tab
    - Version 2.0.x
  - Directory Security Tab
    - Configure SSL Certificate
    - Authentication and Access
      - Disable Anonymous
      - Enable Basic Auth
- Configure Authorization Store Path
  - Edit web.config in DWS directory
  - Change AZ_STORE_PATH value to FULL path of *dwsroot*\App_Data\DWSAzRoles.xml
- [Optional] Constrained DWS access
  - Access to DWS can be constrained beyond the built-in application authorization model by editing the web.config and changing the <authorization> tags. This can limit accounts from accessing DWS altogether.

Assign/Authorize DWS application global administrator

DWS will automatically assign global administrator role to ‘builtin\administrators’ group of the *webserver*. To assign another account this access so they can begin to use DWS, follow:

On *webserver*, start→run→ azman.msc

Right click Authorization Manager → Open Authorization Store

Browser to *dwsroot*\App_Data\DWSAzRoles.xml, open

DWSAzRoles.xml → DWS → Role Assignment → GlobalAdministrator, Right click add new account. This account can be user or group in *domain* or *localhost*

Recycle DWSAppPool in IIS MMC after change

Verify DWS

https://localhost/dwsdir/DhcpSecurity.asmx

Multiple DHCP Servers

DWS can support the management of multiple DHCP servers by adding *serviceaccount* to another DHCP server’s local computer group ‘DHCP Administrators’. The same action completed in step 3 of section ‘Setup Service Account’.

Consume

User accounts that either belong to the GlobalAdministrators application group or builtin\administrators group of *webserver* can begin to use DWS clients to consume the web services. See DWS Management Console client (MMC 3.0) documentation and Perl client samples on DWS media.

APPENDIX C

DHCP Web Services MMC Documentation

Requirements

- Microsoft .NET 2.0
- Microsoft Management Console (MMC) 3.0
  - MMC 3.0 is native for Windows Server 2003 R2 and Vista
  - MMC 3.0 for Windows Server 2003 and XP
    - http://support.microsoft.com/?kbid=907265

Installation

Run install.bat found in the same directory as this document. To start DWS MMC, double-click the Microsoft Common Console Document, dws.msc. Custom consoles can be created by making a new MSC file:

- Start → Run → MMC
- File → Add/Remove Snap-in…
- Add, DHCP Web Services Console

DWS MMC can be uninstalled by running uninstall.bat.

Usage

DWS MMC allows users to consume DHCP Web Services from a Windows OS platform. DWS MMC can support multiple user 'profiles' within a console MSC file. When new profiles and servers are added, the MMC will save the associated connection data within its MSC file. The DWS MMC can manage any DHCP server accessible through DWS.

Profiles

A profile contains a user account ID and a DHCP Web Services connection URL. This allows DWS MMC the ability to support multiple user accounts within one console document. Once a profile is created, it will be saved to the MSC file when the DWS MMC is closed, thus allowing profile data to be saved across console shutdowns. To create a new DWS profile:

- Right click the DWS root node and select 'Create Profile…'
  - Profile Name: Identifying name of profile
  - DWS URL: The directory URL of DHCP Web Services, for example, the value of URL would be https://server/dws/, if the URL path containing DhcpOperations.asmx and DhcpSecurity.asmx files.
  - Username: User account authorized to use DWS
  - Domain: User's Active Directory domain

DHCP Servers

Once a profile is created, management of one or more DHCP server can begin. The set of DHCP servers DWS can manage depends on the installation and setup of DWS itself (See DWS-Install document). Also, the user account assigned in the profile must have some authorizations set for the DHCP Server (See Permissions section); otherwise the user may not see any subnets once they add a DHCP server to the MMC. To add a DHCP server:

- Right click profile node, 'Manage DHCP Server…'
- DHCP Server IP: IP address of DHCP server to manage

Server-wide Dynamic DNS configurations can be changed by right clicking a server node and selecting properties.

Search the DHCP server's client lease database: right click a server node and select 'Search…'
DHCP servers added to the DWS MMC will be saved to the MSC file, allowing servers to be saved across console shutdowns.

Subnets
DHCP subnets are networks in which connected network devices (DHCP clients) are served IP and network configurations from DHCP server. To create a new subnet:

Right click server node, 'Create Subnet…'
Name: A name for subnet
Description: A description of subnet
Range
The range of IP address belonging to subnet will be served IP and network configurations
Start IP: Start IP address of range that the DHCP server will manage
End IP: End IP address of range that the DHCP server will manage
Mask: Network mask of subnet
Lease Time: Lease time duration of dynamic clients
State: Refers to state of service for subnet. Enabled – Subnet is serving client, Disabled – Subnet is not serving clients

DHCP subnets can be changed by right clicking the subnet node and selecting properties. Within properties, Dynamic DNS (DNS tab) and BOOTP (Advanced tab) can be changed.
Search the subnet's client lease database: right click a subnet node and select 'Search…'
Address Pools
Address pools node contains a pool that makes up the subnet's default IP range and zero or more exclusion pools. The default pool can be type Dhcp, Bootp, or Both, configured in the subnet properties advance tab. Exclusion pools are address ranges within the subnet and mark exclusion IP addresses that should not be served leases by the DHCP server. To create a new pool:

Right click Address Pools node, select 'New Exclusion Range…'
Start IP: Start IP address of exclusion pool
End IP: End IP address of exclusion pool

Leases
Leases node is mostly a read only node, showing information about client leases that have been established within the network via DHCP. A client lease can be deleted by right clicking the client lease and selecting delete. This node will also show reservations that have been made in the same DHCP subnet and whether the reservation is active or not.

Reservations
The reservation node shows the current IP reservations made within the DHCP subnet. To create a new reservation:

Right click Reservations node, select 'Add Reservation…'
Name: Name identifier of client, usually the client's hostname
IP Address: IP address that should be assigned to client
MAC Address: Hardware address that identifies client to DHCP server
Description: A description for client
Client Type: DHCP, BOOTP, or Both allowed client type for this reservation

An IP reservation can be changed by right clicking the reservation and selecting properties.

DHCP Options
DHCP options are network related configurations that are sent to a DHCP client upon receiving a lease from the DHCP server. DHCP options can be assigned values at three levels of the DHCP server and are hierarchic. The three levels are server, subnet, and reservation. If an option's value is set at the server level, then subnets and reservations under that server will receive the same option value. Option values can be set via the Options node under the server or subnet nodes in the MMC. To set a reservation's options, right click the reservation, select properties, and chose the Options tab.
Some common option and values type are:

DHCP Option	Data Type	Example values
003 – Router	Array of IP addresses	192.168.0.1
006 – DNS Servers	Array of IP addresses	192.168.0.5, 192.168.0.6
015 – DNS Domain Name	String	unf.edu

See http://www.faqs.org/rfcs/rfc2132.html for more information on DHCP options.
There is also a concept of option classes, which includes three types: DHCP, BOOTP, and Routing and Remote clients. An option can be assigned a value for a specific type of client class. For example, if the router option was set within the BOOTP class. Then only BOOTP client types would receive that router option value when receiving a lease from DHCP server.

Permissions and Security Trimming
Users whom have been granted global administrator rights within DWS will see permission nodes throughout the DWS MMC tree. Permission nodes allow administrators to assign various authorizations to sections of the DHCP server to users. Those users can then manage that section (or scope) of the DHCP server.

Authorizations are enforced using two methods. The first is a direct allow/deny method. If a user does not have sufficient authorization access to complete an operation, like CreateSubnet, the operation will throw an unauthorized exception. Otherwise, the operation will allow the creation to occur. The second method is security trimming. In this approach, DHCP operations that return a set of DHCP objects are 'trimmed' to exclude objects to which the user does not have access. For example, if a user has read access to a range of IP addresses within a subnet, and executes the EnumClients operation, the operation would normally return all client leases within that subnet. With security trimming, however, the caller will only receive client leases within the range of IP addresses to which he has been granted read access. Furthermore, the set of client leases could be empty if the user does not have access to read any leases within a subnet.

Searching
Searching allow users to query a DHCP server for client leases based on regular expression patterns. The search filter allows three terms to be used in a query: client hostname, client IP address, and client MAC address. The search filter matches based on a logical AND condition of non-empty terms. The regular expression patterns should follow the .NET regular expression language definition, http://msdn2.microsoft.com/en-us/library/az24scfc(VS.80).aspx.
Examples:

Terms	Patterns	Results
IP: Mac: Name:	^192.168.2.*$ *D8$	Client leases that contain first 3 bytes 192.168.2 AND their MAC address ends with byte D8
IP: Mac: Name:	 ^*.unf.edu$	Client leases containing hostnames ending in unf.edu

APPENDIX D

Perl Client Samples

```
#!/usr/bin/perl
#
# DHCP Web Services Client
# using SOAP::Lite
#
# Creates a DHCP Reservation
#
# (c) Jason Rupard, School of Computing, University of North Florida
#

#use SOAP::Lite +trace => debug;
use SOAP::Lite;

my $dws = SOAP::Lite
  -> uri('http://www.unf.edu/~jrupard/dws/')
  -> proxy('https://jr1/x/DhcpOperations.asmx')
  -> on_action( sub { join '/', 'http://www.unf.edu/~jrupard/dws', $_[1] } );

my $serverIp = '192.168.0.5';
my $networkIp = '192.168.200.64';
my $resIp = '192.168.200.70';
my $resMac = '0000302010A0';
my $clientName = 'MyClientResHostName';

my $serverXML = SOAP::Data->name(
       server => \SOAP::Data->value(SOAP::Data->name(Ip => $serverIp))
    );
my $subnetXML = SOAP::Data->name(
       network => \SOAP::Data->value(SOAP::Data->name(Ip => $networkIp ))
    );

my $rezXML =
 SOAP::Data->name(res => \SOAP::Data->value(
   SOAP::Data->name(ReservedIp => \SOAP::Data->value(
      SOAP::Data->name(Ip => $resIp))),
   SOAP::Data->name(ReservedMac => \SOAP::Data->value(
       SOAP::Data->name(Mac => $resMac))),
   SOAP::Data->name(bAllowedClientTypes => 'Dhcp'),
   SOAP::Data->name(Name => $clientName),
   SOAP::Data->name(Comment => 'Made via SOAP::LITE')
 ));

my $res = $dws->CreateReservation($serverXML, $subnetXML, $rezXML);
die $res->faultstring."\n" if  ($res->fault);


sub SOAP::Transport::HTTP::Client::get_basic_credentials {
  return $user => $pass;
}

#!/usr/bin/perl
#
# DHCP Web Services Client
# using SOAP::Lite
#
# Creates a DHCP Subnet and sets DHCP Option 51 (Lease Time)
#
```

```
# (c) Jason Rupard, School of Computing, University of North Florida
#
#

#use SOAP::Lite +trace => debug;
use SOAP::Lite;

my $dws = SOAP::Lite
  -> uri('http://www.unf.edu/~jrupard/dws/')
  -> proxy('https://jr1/x/DhcpOperations.asmx')
  -> on_action( sub { join '/', 'http://www.unf.edu/~jrupard/dws', $_[1] } );

my $serverIp = '192.168.0.5';
my $networkIp = '192.168.200.64';
my $networkMask = '255.255.255.192';
my $startIp = '192.168.200.65';
my $endIp = '192.168.200.126';

#Server Info
my $serverXML = SOAP::Data->name(
        server => \SOAP::Data->value(SOAP::Data->name(Ip => $serverIp))
    );

#Subnet Info
my $subnetXML =
 SOAP::Data->name(network => \SOAP::Data->value(
     SOAP::Data->name(Address => \SOAP::Data->value(SOAP::Data->name(Ip =>
$networkIp))),
     SOAP::Data->name(Mask => \SOAP::Data->value(SOAP::Data->name(Ip =>
$networkMask))),
     SOAP::Data->name(Name => 'Made from Perl::Lite'),
     SOAP::Data->name(Comment => 'w00t!'),
     SOAP::Data->name(State => 'Disabled')
     )
 );

#Range Info
my $rangeXML =
 SOAP::Data->name(ipRange => \SOAP::Data->value(
     SOAP::Data->name(Type => 'DhcpIpRangesDhcpOnly'),
     SOAP::Data->name(StartAddress => \SOAP::Data->value(SOAP::Data->name(Ip =>
$startIp))),
     SOAP::Data->name(EndAddress => \SOAP::Data->value(SOAP::Data->name(Ip =>
$endIp)))
     )
 );

#Lease Time
my $optValue51XML =
 SOAP::Data->name(value => \SOAP::Data->value(
     SOAP::Data->name(OptionId => 51)->type('unsignedInt'),
     SOAP::Data->name(ClassType => 'Dhcp'),
     SOAP::Data->name(OptionData => \SOAP::Data->value(
         SOAP::Data->name(Type => 'DWordType'),
         SOAP::Data->name(Data => \SOAP::Data->value(
             SOAP::Data->name(unsignedInt => 3660))
         )
     ))->attr({ 'xsi:type' => "DhcpOptionDataDword" }),
     SOAP::Data->name(OptionScope => \SOAP::Data->value(
         SOAP::Data->name(type => 'Subnet'),
         SOAP::Data->name(SubnetIp => \SOAP::Data->value(
             SOAP::Data->name(Ip => $networkIp))
         )
     ))
 ));

my $res = $dws->CreateSubnet($serverXML, $subnetXML, $rangeXML);
 die $res->faultstring."\n" if  ($res->fault);
 $res = $dws->SetOptionValue($serverXML, $optValue51XML);
 die $res->faultstring."\n" if  ($res->fault);


sub SOAP::Transport::HTTP::Client::get_basic_credentials {
  return $user => $pass;
}

#!/usr/bin/perl
#
# DHCP Web Services Client
# using SOAP::Lite
#
# Enumerates Subnets on DHCP Server
#
# (c) Jason Rupard, School of Computing, University of North Florida
#
#

#use SOAP::Lite +trace => debug;
use SOAP::Lite;

my $soap = SOAP::Lite
  -> uri('http://www.unf.edu/~jrupard/dws/')
  -> proxy('https://jr1/x/DhcpOperations.asmx')
  -> on_action( sub { join '/', 'http://www.unf.edu/~jrupard/dws', $_[1] } );

my $serverIp = SOAP::Data->name(
        server => \SOAP::Data->value(SOAP::Data->name(Ip => '192.168.0.5')));

my $res = $soap->EnumSubnets($serverIp);
die $res->faultstring."\n" if not  ($res || $res->fault);

if($res->result->{DhcpSubnet} =~ /ARRAY/) {
 @subnets = @{$res->result->{DhcpSubnet}};
}
else {
 @subnets = ($res->result->{DhcpSubnet});
}

printf "\n%-24s %-12s %-12s %-10s\n", "SubnetName", "Network", "Mask", "State";
printf "%-24s %-12s %-12s %-10s\n", "-"x24, "-"x12, "-"x12, "-"x10;
foreach(@subnets)
{
 printf "%-24s %-12s %-12s %-10s\n", $_->{Name}, $_->{Address}->{Ip}, $_->{Mask}-
>{Ip}, $_->{State};
}

print "\n";

sub SOAP::Transport::HTTP::Client::get_basic_credentials {
  return $user => $pass;
}

#!/usr/bin/perl
#
# DHCP Web Services Client
# using SOAP::Lite
#
# Search entire DHCP server
#
# Command Line:
#  -i <IP regex pattern>
#  -m <MAC regex pattern>
#  -n <Name regex pattern>
#
# regex is a .NET regular express format and case insensitive
# http://msdn2.microsoft.com/en-us/library/az24scfc(VS.80).aspx
#
# (c) Jason Rupard, School of Computing, University of North Florida
```

```
#
#

#use SOAP::Lite +trace => debug;
use SOAP::Lite;
use Getopt::Std;

my $dws = SOAP::Lite
  -> uri('http://www.unf.edu/~jrupard/dws/')
  -> proxy('https://jr1/x/DhcpOperations.asmx')
  -> on_action( sub { join '/', 'http://www.unf.edu/~jrupard/dws', $_[1] } );

my $serverIp = SOAP::Data->name(
        server => \SOAP::Data->value(SOAP::Data->name(Ip => '192.168.0.5'))
    );

getopts('i:m:n:', \ my %opts);

my $terms =
 SOAP::Data->name(terms => \SOAP::Data->value(
   SOAP::Data->name(IpRegEx => $opts{i} || ''),
   SOAP::Data->name(MacRegEx => $opts{m}|| ''),
   SOAP::Data->name(NameRegEx => $opts{n} || '')
 ));

my $res = $dws->ServerFindAllClients($serverIp, $terms);
die $res->faultstring."\n" if  ($res->fault);


my @clients;
if($res->result->{DhcpClient} =~ /ARRAY/) {
 @clients = @{$res->result->{DhcpClient}};
}
else {
 @clients = ($res->result->{DhcpClient});
}


printf "\n%-24s %-12s %-12s %-10s\n", "IpAddress", "Mac", "Name", "LeaseExpires";
printf "%-24s %-12s %-12s %-10s\n", "-"x24, "-"x12, "-"x12, "-"x10;
foreach(@clients)
{
 printf "%-24s %-12s %-12s %-10s\n", $_->{IpAddress}->{Ip}, $_->{MacAddress}->{Mac},
$_->{Name}, $_->{LeaseExpires};
}

printf "%-24s %-12s %-12s %-10s\n", "-"x24, "-"x12, "-"x12, "-"x10;
printf "%-24s %-12s %-12s %-10s\n", "IpAddress", "Mac", "Name", "LeaseExpires";
print "\n";

sub SOAP::Transport::HTTP::Client::get_basic_credentials {
  return $user => $pass;
}
```

VITA

Jason Rupard has a Bachelor of Science in Computer and Information Sciences from the University of North Florida, 2002, and expects to receive a Master of Science in Computer and Information Sciences from the University of North Florida, May 2008. Dr. Yap S. Chua of the University of North Florida is serving as Jason's project director. Jason has been employed since 2005, by the University of North Florida as a systems engineer in the Information Technology Services department. Jason's previous jobs included a graduate assistantship with the University of North Florida and a graduate fellowship with the Mayo Clinic. While attending the University of North Florida, Jason fulfilled several roles of the UNF Chapter of the Association for Computing Machinery (ACM), from president to web master. Additionally, he was inducted into the UNF chapters of the honor societies, Pi Mu Epsilon and Upsilon Pi Epsilon. Jason also worked five years, while in college, with Outback Steakhouse as a cook.

Jason's interests include system engineering and system development. He has experience engineering Linux and Windows server system solutions, and developed major academic projects using C, C++, C#, and Java. Jason has used C, C#, PERL and SQL to develop solutions, during his professional career. He enjoys computer games and administrating on-line game communities. Jason hopes to be a major contributor to an open-source project in the future. Married for the last 2.5 years, Jason has one child, William, age 7 months.