NETAPP MANAGEABILITY SDK C# PROGRAMMING FOR BEGINNERS

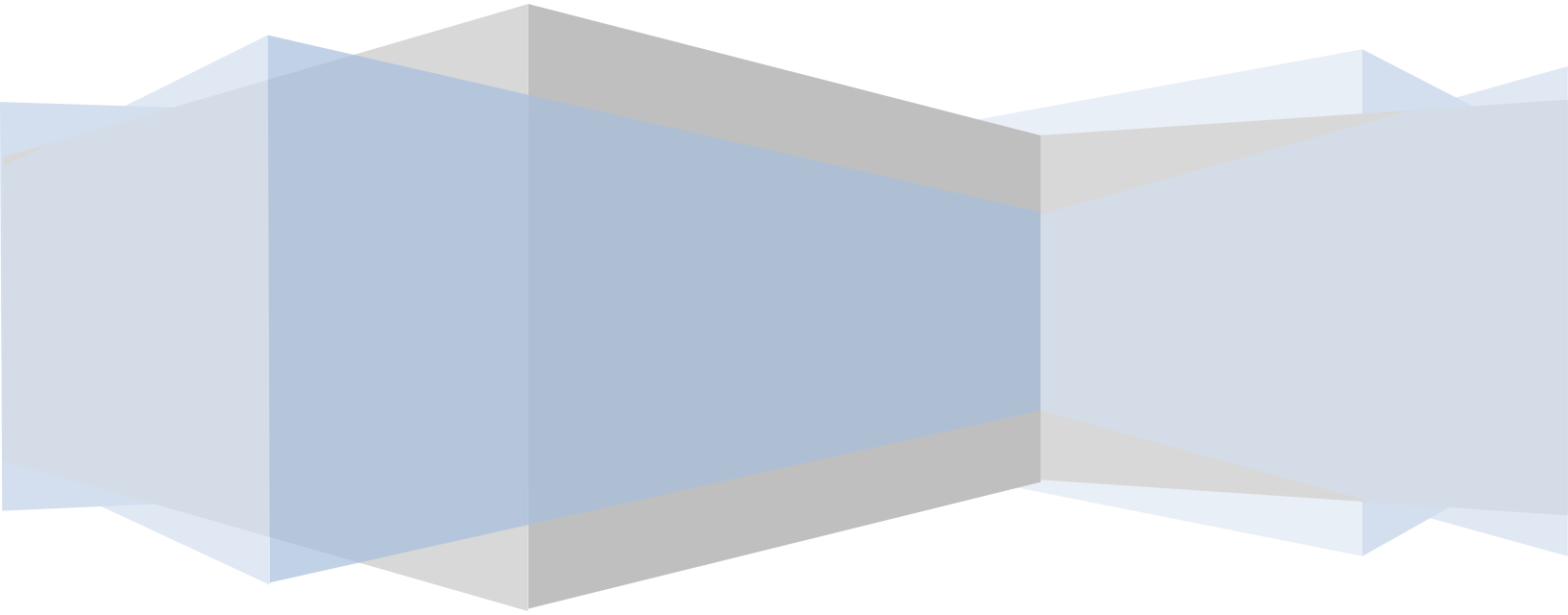


TABLE OF CONTENTS

SOFTWARE REQUIREMENTS:

- Manage ONTAP SDK 3.5 or NetApp Manageability SDK 4.0
- Microsoft .NET Framework 2.0, 3.0 or 3.5
- This document makes use of Microsoft Visual C# 2008 Express Edition. This is not a mandatory requirement.

INTRODUCTION

SDK CORE APIS

The SDK core APIs provide infrastructure to invoke the Data ONTAP APIs on a storage system. The SDK Core APIs provide the client side interfaces to set up connection with the storage system, pack and unpack messages in the XML format, query the Data ONTAP APIs on the storage system, determine the queried result, and so on. The data structures used for sending requests and for receiving responses are called Input Output (I/O) elements. XML encoding is used to package these I/O elements. Each element has a name and content, and might have one or more child elements within. The content type is a string, integer, array of elements, or a NetApp-defined data type.

The SDK Core APIs internally convert the Data ONTAP API to be invoked, and the I/O elements, to XML format. The mechanism is as follows:

- The client code uses a small set of primitives to create a request.
- The only primitive data types available in the SDK Core APIs are integer, string, and boolean. Nested structures and arrays are also available, which contain integer, string, and boolean, in turn.
- An API name is associated with it.
- Named parameters are added to the request.

On the server side, a similar set of primitives extracts the parameters from the request for processing, and packages the reply. At the client side, the response data is extracted using the SDK Core APIs. The transport mechanism used for the API communication is HTTP, HTTPS, or DCE/RPC for Windows. HTTP and HTTPS are important while managing devices residing outside of the corporate firewall. HTTPS provides secure communication by encrypting the API requests and responses.

For complete documentation of the core APIs and Data ONTAP APIs, please refer the documentation in NetApp Manageability SDK bundle. Below is the list of main classes implementing core APIs in C#.

Class	Description
NaServer	This class provides the methods to set up server connection and session parameters, send Data ONTAP APIs to the storage system, parse the API response, and so on.

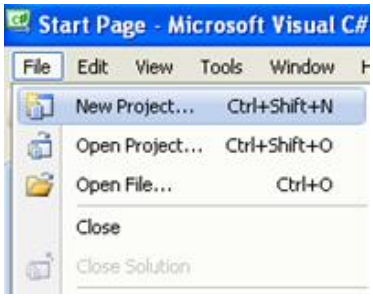
NaElement	This class encapsulates the input output (I/O) elements. An NaElement class encapsulates one level of an I/O element. You can nest these I/O elements arbitrarily. They have names, corresponding to the XML tags, attributes (only used for results), values (always Strings), and children, corresponding to the nested tagged items. The methods of this class are used to package the requests and responses, create new elements, change the contents of the elements, and so on.
NaException	This is the base class for all the ONTAPI exceptions. It is thrown when there is a problem invoking an API.
NaApiFailedException	This exception indicates a failed API invocation. The server processed the request and returned a failure result including an error no and a reason for failure in string.
NaAuthException	This exception is thrown when the remote server does not accept the credentials provided with the request, or when the required credentials are missing.
NaConnectionException	This exception is thrown when there is an error connecting to the remote server.
NaProtocolException	This exception is thrown when the response of the remote server is not known.

DEVELOPMENT ENVIRONMENT SETUP

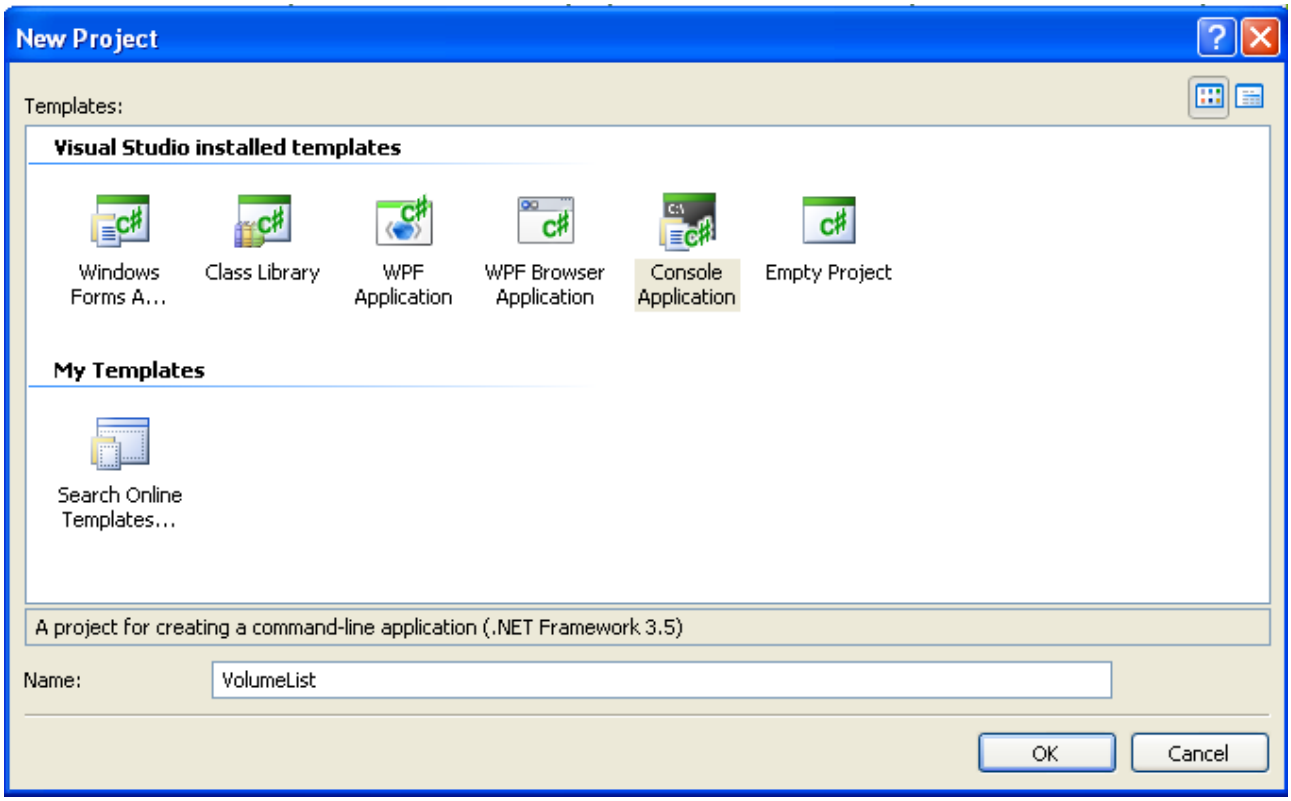
For this document, Microsoft Visual C# 2008 Express Edition is used as the development environment. Let's try to understand how to go about writing a C# program using the .NET library provided in NetApp Manageability SDK.

PREPARE ENVIRONMENT

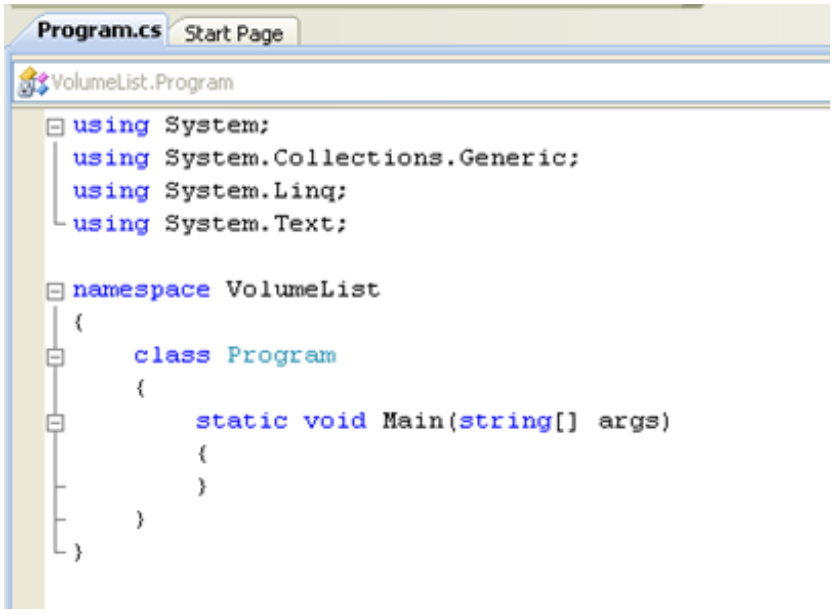
Let's write a program to query all volumes on the storage system. To start with, after launching the Visual C# Express Edition console, create a new project.



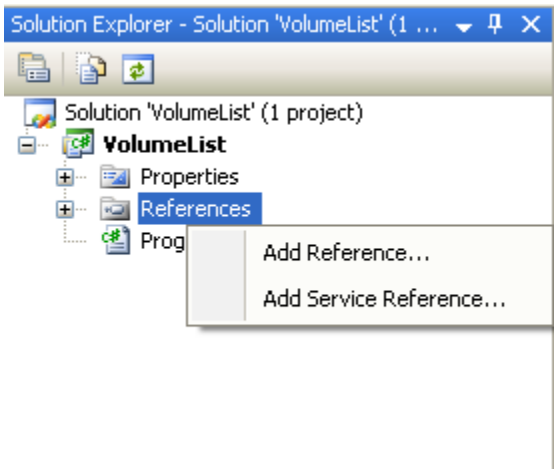
Choose a console application and name your project “VolumeList”.



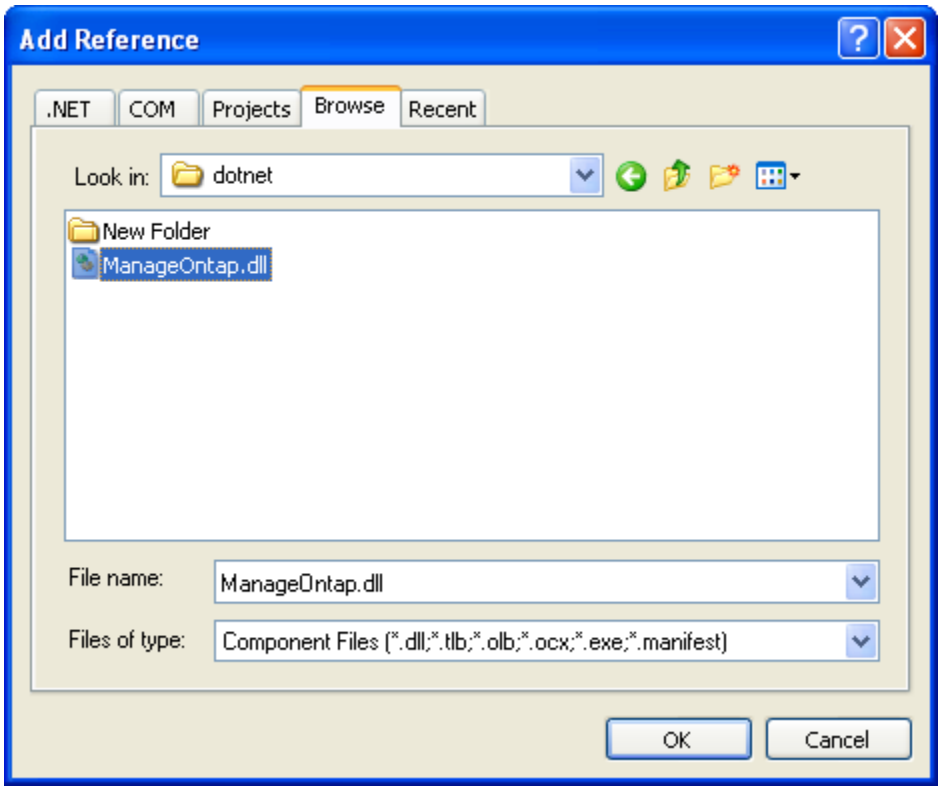
Once you click “OK”, you will get a skeleton program structure as shown in the figure below.



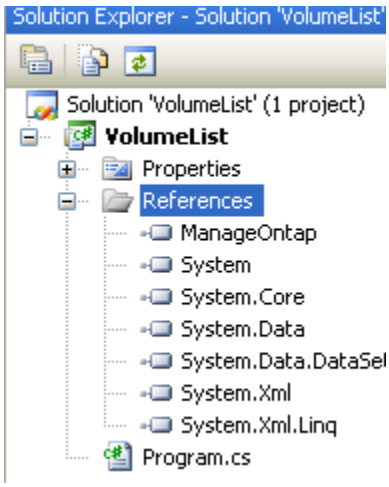
Now, in order to make use of the .NET library provided in the SDK, right click on the “References” and “Add Reference”.



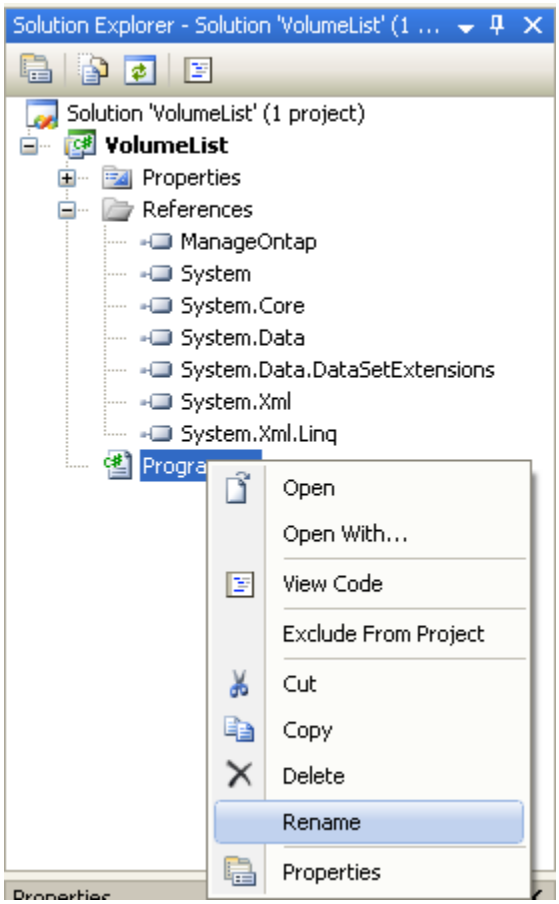
When prompted to add a new reference, choose the “Browse” tab and navigate to “<SDK DIR>\lib\DotNet\ManageOntap.dll”.

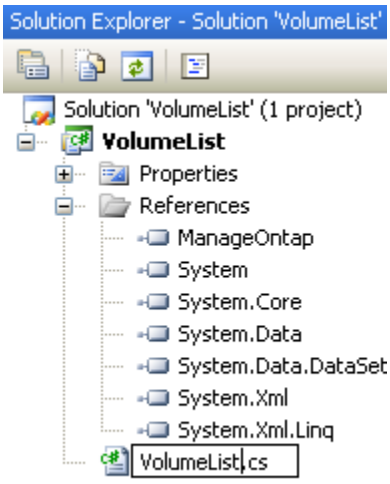


The “Solution Explorer” tab will now display “ManageOntap” reference.

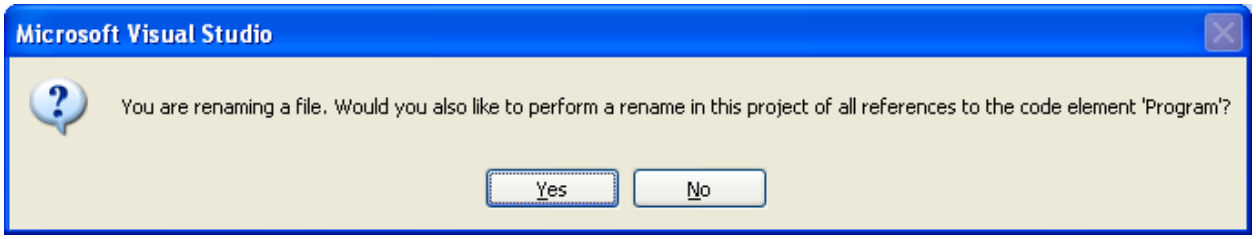


If needed rename the C# program file name to be in line with the project name as shown below.

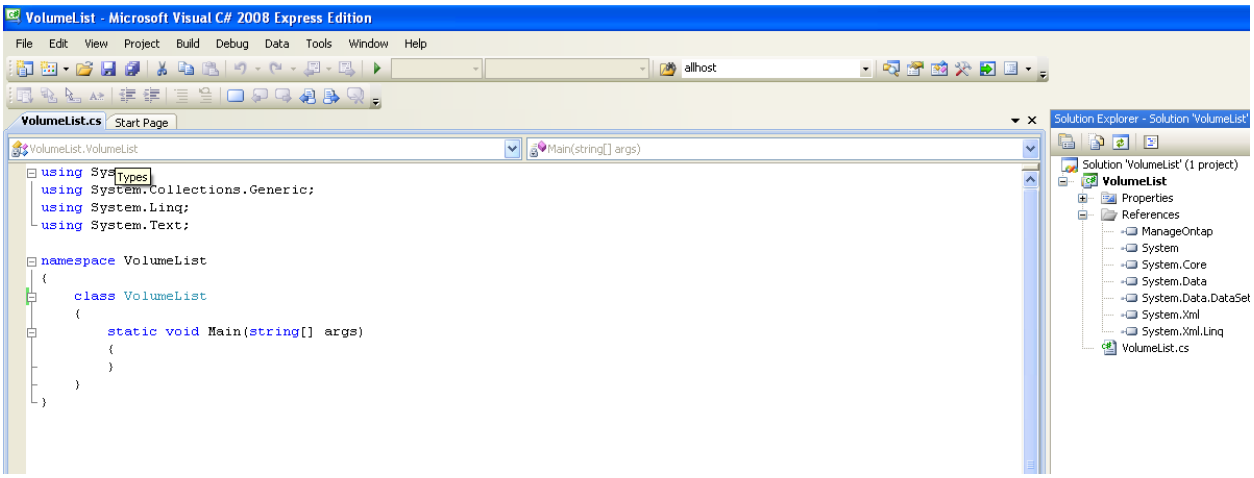




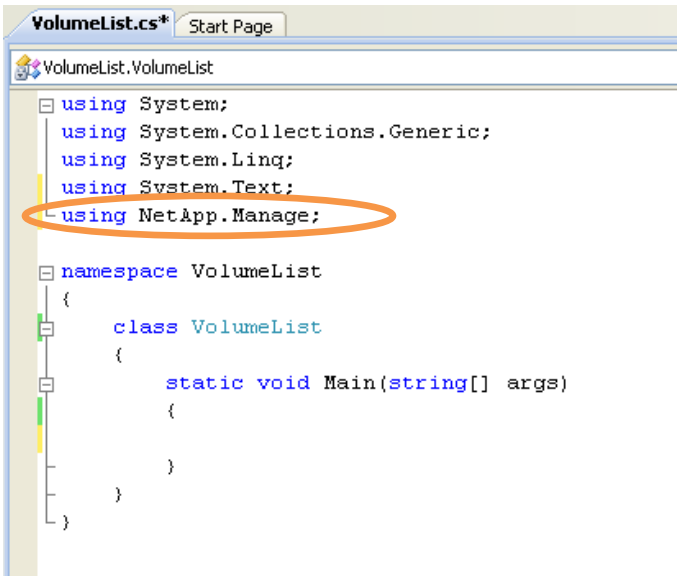
Choose “Yes” when prompted.



The setup is now complete for you to go ahead and write programs in C# to interact with Data ONTAP on NetApp Storage Systems.



Add the “using” directive to enable use of NetApp namespace.



WRITE A C# PROGRAM

Let's go over a program to understand how to invoke Data ONTAP APIs using the core APIs in SDK.

First of all, add NetApp namespace via "using" directive.

```
Using NetApp.Manage;
```

```
namespace VolumeList
{
    class VolumeList
    {
        static void Main(string[] args)
        {
```

Declare a Server Object, which is needed to set server context on which the Data ONTAP API is to be invoked.

```
NaServer  s;
```

Declare a NaElement input object to hold Data ONTAP API.

```
NaElement xi;
```

Declare a NaElement output object to hold the output returned from API call.

```
NaElement xo;
```

Logic to make sure you input the right number of command line arguments

```
if ( args.Length < 3 ) {
 Console.WriteLine("Usage:VolumeList <filername> <user> <passwd>[<volume>]");
 System.Environment.Exit(1);
}
```

Assign command line inputs to variables.

```
String Server = args[0], user = args[1], pwd = args[2];

try {

  Console.WriteLine("|-------------------------------------|");
  Console.WriteLine("|Program to Demo use of Volume APIs to list Volume info
|");
  Console.WriteLine("|-------------------------------------|");
```

The first thing a client program must do is initialize the server context. In this example, you create a server object "s" identifying that you would want to talk to NetApp Storage System named "Server" with Data ONTAP API version 1.3. You can as well provide the IP address here. Additionally, set how you would want to authenticate with the storage system, in this case using login/password. Since the authentication style is login/password, set the user credentials. For other options that can be set for server context refer SDK documentation section for "NaServer" class and its members.

```
//Initialize connection to server, and
//request version 1.3 of the API set
//
s = new NaServer(Server,1,3);
s.Style = NaServer.AUTH_STYLE.LOGIN_PASSWORD;
s.SetAdminUser(user, pwd);
s.TransportType = NaServer.TRANSPORT_TYPE.HTTP;
```

Now that you have defined the server context on the client, next define the input API that needs to be sent across to the server. In this example, we are trying to query the list of volumes on the storage system. The API to do this is "volume-list-info". Let's take a look at the API from the SDK documentation.

volume-list-info

Get volume status. Note that all RAID-related status items (e.g., 'raid-size', 'raid-status', 'checksum-style') reported for a flexible volume actually describe the state of its containing aggregate.

Input Name	Type	Description
verbose	boolean optional	If set to "true", more detailed volume information is returned. If not supplied or set to "false", this extra information is not returned.
volume	string optional	The name of the volume for which we want status information. If not supplied, then we want status for all volumes on the filer.

From the documentation it is clear that the API "volume-list-info" takes two arguments – verbose and volume, both being optional. Hence to gather details of all volumes we need not specify the volume argument. In case we need details of a specific volume we have to provide the volume argument. Lets consider both cases in our program.

To create an input object, use NaElement class. To get details on a specific volume, add the volume name as an argument to the input object, using "AddNewChild" method. In example below, we first create an input element using NaElement. Next we check the number of command line arguments. If the number of arguments is 4, we assume the last argument is the volume name that is of interest. Hence add the volume argument using "AddNewChild" method.

```
xi = new NaElement("volume-list-info");
if(args.Length==4){
      String volume_name = args[3];
      xi.AddNewChild("volume",volume_name);
}
```

Now that the input element is ready, this needs to be passed to the server to be executed. To achieve this we have to use the "InvokeElem" method. Below is the example of how to do it.

```
//Invoke Vol list Info ONTAPI API
xo = s.InvokeElem(xi);
```

If the API query has succeeded, the output is stored in object "xo" in XML format. The next thing to do is understand how to extract data from the returned object "xo". Before we do that, let us have a look at what is to be returned from the API call. Below table shows that the API call will return output as a list of volumes. All details regarding volumes that is returned will be in an XML format, hence needs to be extracted using the core SDK APIs.

Output Name	Type	Description
volumes	volume-info[]	List of volumes and their status information.

Since the returned output is an array of volume-info elements, we need to iterate though the list and print relevant details that are of interest to us. Below example shows how to iterate and extract data using "GetChildContent" method.

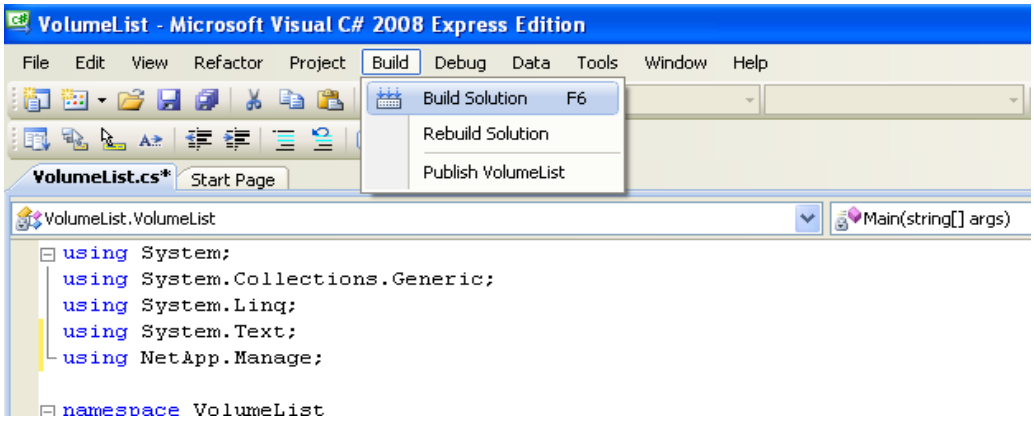
```
//Get the list of children from element(Here 'xo') and iterate
//through each of the child element to fetch their values
//

System.Collections.IList volList =xo.GetChildByName("volumes").GetChildren();
```

```
System.Collections.IEnumerator volIter = volList.GetEnumerator();

while(volIter.MoveNext()){
    NaElement volInfo=(NaElement)volIter.Current;

    Console.WriteLine("---------------------------------");
    Console.Write("Volume Name\t\t: ");
    Console.WriteLine(volInfo.GetChildContent("name"));

    Console.Write("Volume State\t\t: ");
    Console.WriteLine(volInfo.GetChildContent("state"));

    Console.Write("Disk Count\t\t: ");
    Console.WriteLine(volInfo.GetChildIntValue("disk-count",-1));

    Console.Write("Total Files\t\t: ");
    Console.WriteLine(volInfo.GetChildIntValue("files-total",-1));

    Console.Write("No of files used\t: ");
    Console.WriteLine(volInfo.GetChildIntValue("files-used",-1));

    Console.WriteLine("---------------------------------");
}
```

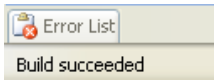
If the API has fails, exceptions can be caught and relevant error messages can be printed on the console.

```
      catch (Exception e) {
                Console.Error.WriteLine(e.Message);
      }
```

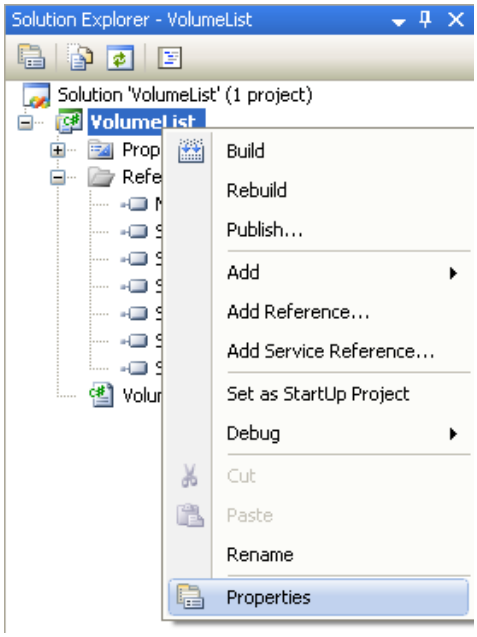
The basic program is now complete. Add more as needed. To compile, just use the Build Solution from Build menu.



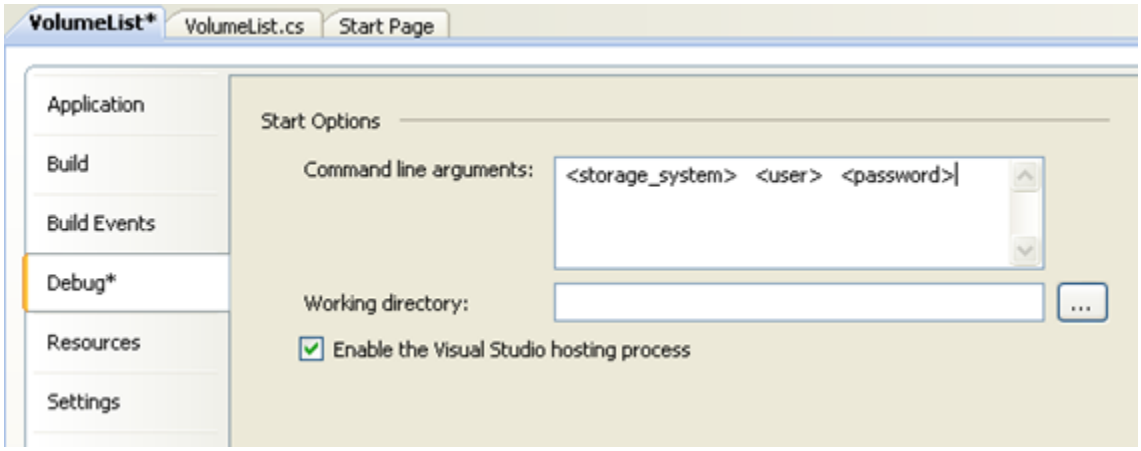
If the build process completes without errors, you'll see the below success message at the bottom left corner of the window.



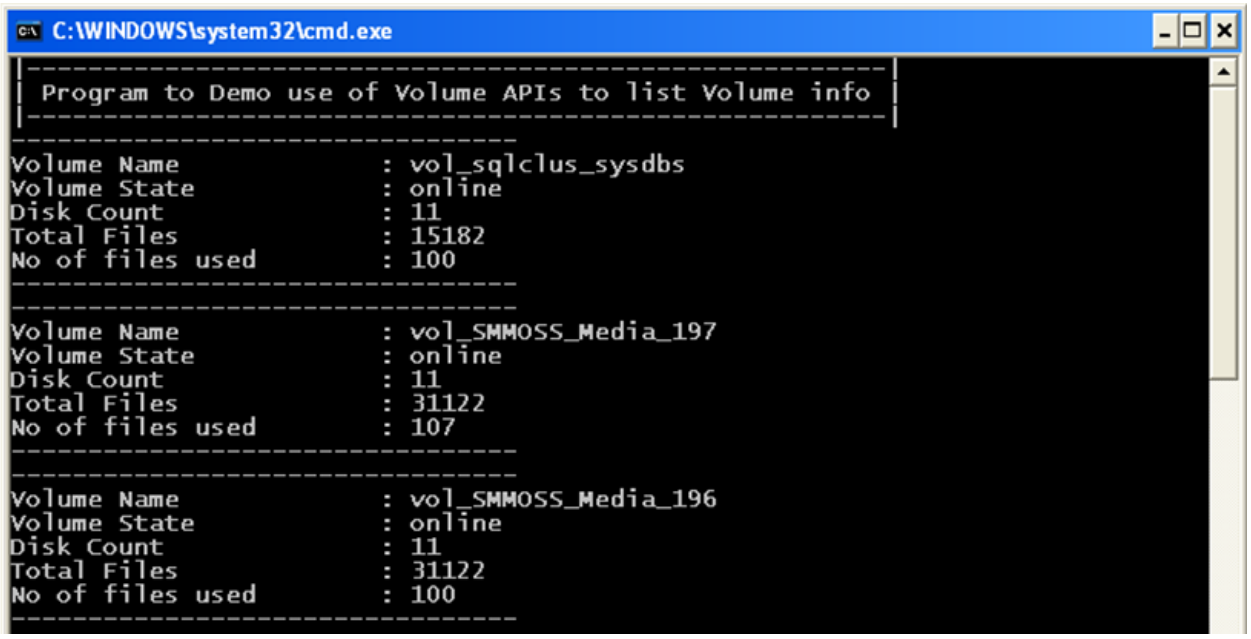
Before you try to run the program to check or debug, make sure you provide the command line arguments as needed. For our example you can provide the command line arguments as shown below.



Right click on the solution name "VolumeList" as shown above to get the below property window.



To execute the program, press Ctrl+F5.



APPENDIX – SAMPLE PROGRAMS

LIST ALL OR SPECIFIC VOLUME DETAILS ON A STORAGE SYSTEM

```
//===================================================================//
// Usage: vollist  <filer> <user> <password> [volume]                //
//===================================================================//
using System;
using System.Collections.Generic;
using System.Text;
using NetApp.Manage;

namespace vollist
{
    class VolList
    {
```

```
        static void Main(string[] args)
        {
                  NaElement xi;
                NaElement xo;
                NaServer  s;

                if ( args.Length < 3 ) {
                      Console.WriteLine("Usage    :    vollist    <filername>
<username> <passwd> [<volume>]");
                System.Environment.Exit(1);
                }

            String Server = args[0], user = args[1], pwd = args[2];

            try {

                Console.WriteLine("|----------------------------------------
---------------|");
                Console.WriteLine("| Program to Demo use of Volume APIs to
list Volume info |");
                Console.WriteLine("|----------------------------------------
---------------|");

                      //Initialize connection to server, and
                      //request version 1.3 of the API set
                      //
                      s = new NaServer(Server,1,3);
                      s.Style = NaServer.AUTH_STYLE.LOGIN_PASSWORD;
                      s.SetAdminUser(user, pwd);

                //Create Vol list Info ONTAPI API
                xi = new NaElement("volume-list-info");
                      if(args.Length==4){
                     String volume_name = args[3];
                            xi.AddNewChild("volume",volume_name);
                      }

                //Invoke Vol list Info ONTAPI API
                  xo = s.InvokeElem(xi);
                      //
                      //Get the list of children from element(Here 'xo') and
iterate
                      //through each of the child element to fetch their
values
                      //
                System.Collections.IList              volList              =
xo.GetChildByName("volumes").GetChildren();
                System.Collections.IEnumerator           volIter           =
volList.GetEnumerator();
                      while(volIter.MoveNext()){
                            NaElement volInfo=(NaElement)volIter.Current;
                            Console.WriteLine("----------------------------
---");
                            Console.Write("Volume Name\t\t: ");

Console.WriteLine(volInfo.GetChildContent("name"));
                    Console.Write("Volume State\t\t: ");
```

```
Console.WriteLine(volInfo.GetChildContent("state"));
                    Console.Write("Disk Count\t\t: ");
                            Console.WriteLine(volInfo.GetChildIntValue("disk-
count",-1));
                    Console.Write("Total Files\t\t: ");

Console.WriteLine(volInfo.GetChildIntValue("files-total",-1));
                    Console.Write("No of files used\t: ");

Console.WriteLine(volInfo.GetChildIntValue("files-used",-1));
                    Console.WriteLine("---------------------------------");
                  }
            }
                catch (Exception e) {
                Console.Error.WriteLine(e.Message);
                }
          }
    }
}
```

SNAPSHOT MANAGEMENT

```
using System;
using System.Collections.Generic;
using System.Text;
using NetApp.Manage;

namespace snapman
{
    class SnapMan
    {
        static void Usage(string[] args)
        {
            Console.WriteLine(
                    "Usage: snapman -g <filer> <user> <pw> <vol> \n" +
                    "               -l <filer> <user> <pw> <vol> \n" +
                    "                          -c  <filer>  <user>  <pw>  <vol>
<snapshotname> \n" +
                    "                          -r  <filer>  <user>  <pw>  <vol>
<oldsnapshotname> <newname> \n" +
                    "                          -d  <filer>  <user>  <pw>  <vol>
<snapshotname> \n\n");
          Console.WriteLine("E.g.  snapman  -l  filer1  root  6a55w0r9  vol0
\n\n");
            Console.WriteLine(
                    "Use -g to get the snapshot schedule\n" +
                    "    -l to list snapshot info \n" +
                    "    -c to create a snapshot \n" +
                    "    -r to rename one \n" +
                    "    -d to delete one \n");

            System.Environment.Exit(-1);
        }
```

```
        static void CreateSnapshot(String[] args)
        {
            NaServer s;
            string filer = args[1];
            string user = args[2];
            string pwd = args[3];
            string vol = args[4];
            string ssname = args[5];
            NaElement xi, xo;

            try
            {
                s = new NaServer(filer, 1, 0);
                s.Style = NaServer.AUTH_STYLE.LOGIN_PASSWORD;
                s.TransportType = NaServer.TRANSPORT_TYPE.HTTP;
                s.SetAdminUser(user, pwd);

                xi = new NaElement("snapshot-create");
                if (args.Length == 6)
                {
                    xi.AddNewChild("volume", vol);
                    xi.AddNewChild("snapshot", ssname);
                }
                else
                {
                    Console.Error.WriteLine("Invalid number of arguments");
                    Usage(args);
                    System.Environment.Exit(-1);
                }

                xo = s.InvokeElem(xi);
                //
                // print it out
                //
                Console.WriteLine("Snapshot " + ssname + " created for volume
" + vol + " on filer " + filer);
            }
            catch (NaException e)
            {
                Console.Error.WriteLine("ERROR: " + e.Message);
            }
        }


        static void DeleteSnapshot(String[] args)
        {
            try
            {
                NaServer s;
                string filer = args[1];
                string user = args[2];
                string pwd = args[3];
                string vol = args[4];
                string ssname = args[5];
                NaElement xi, xo;

                s = new NaServer(filer, 1, 0);
```

```
                s.Style = NaServer.AUTH_STYLE.LOGIN_PASSWORD;
                s.TransportType = NaServer.TRANSPORT_TYPE.HTTP;
                s.SetAdminUser(user, pwd);

                xi = new NaElement("snapshot-delete");
                xi.AddNewChild("volume", vol);
                xi.AddNewChild("snapshot", ssname);

                xo = s.InvokeElem(xi);
                //
                // print it out
                //
                Console.WriteLine("Snapshot  "  +  ssname  +  "  deleted  from
volume " + vol + " on filer " + filer);
            }
            catch (IndexOutOfRangeException e)
            {
                Console.Error.WriteLine("Invalid number of arguments");
                Usage(args);
                Console.Error.WriteLine(e.Message);
                System.Environment.Exit(-1);
            }
            catch (NaException e)
            {
                Console.Error.WriteLine("ERROR: " + e.Message);
            }
        }


        static void RenameSnapshot(String[] args)
        {
            try
            {
                NaServer s;
                string filer = args[1];
                string user = args[2];
                string pwd = args[3];
                string vol = args[4];
                string ssnameOld = args[5];
                string ssnameNew = args[6];
                NaElement xi, xo;

                s = new NaServer(filer, 1, 0);
                s.Style = NaServer.AUTH_STYLE.LOGIN_PASSWORD;
                s.TransportType = NaServer.TRANSPORT_TYPE.HTTP;
                s.SetAdminUser(user, pwd);

                xi = new NaElement("snapshot-rename");
                if (args.Length == 7)
                {
                    xi.AddNewChild("volume", vol);
                    xi.AddNewChild("current-name", ssnameOld);
                    xi.AddNewChild("new-name", ssnameNew);
                }
                else
                {
                    Console.Error.WriteLine("Invalid number of arguments");
```

```
                Usage(args);
                System.Environment.Exit(-1);
            }

            xo = s.InvokeElem(xi);
            //
            // print it out
            //
            Console.WriteLine("Snapshot " + ssnameOld + " renamed to " +
                                        ssnameNew + " for volume " + vol + "
on filer " + filer);
        }
        catch (IndexOutOfRangeException e)
        {
            Console.Error.WriteLine("Invalid number of arguments");
            Usage(args);
            System.Environment.Exit(-1);
        }
        catch (NaException e)
        {
            Console.Error.WriteLine("ERROR: " + e.Message);
        }
    }


    static void ListInfo(String[] args)
    {
        NaServer s;
        string filer = args[1];
        string user = args[2];
        string pwd = args[3];
        string vol = args[4];

        NaElement xi, xo;
        //
        // get the schedule
        //
        try
        {
            s = new NaServer(filer, 1, 0);
            s.Style = NaServer.AUTH_STYLE.LOGIN_PASSWORD;
            s.TransportType = NaServer.TRANSPORT_TYPE.HTTP;
            s.SetAdminUser(user, pwd);

            xi = new NaElement("snapshot-list-info");
            xi.AddNewChild("volume", vol);
            xo = s.InvokeElem(xi);

            System.Collections.IList              snapshots             =
xo.GetChildByName("snapshots").GetChildren();
            System.Collections.IEnumerator           snapiter           =
snapshots.GetEnumerator();
            while (snapiter.MoveNext())
            {
                NaElement snapshot = (NaElement)snapiter.Current;
                Console.WriteLine("SNAPSHOT:");
```

```
                    int acc 粑 sstime = snapshot.GetChildIntValue("access-
time",0);
                    DateTime            datetime            =            new
DateTime(1970,1,1,0,0,0).AddSeconds(accesstime);
                    Console.WriteLine("              NAME     \t\t=     "     +
snapshot.GetChildContent("name"));
                    Console.WriteLine("       ACCESS   TIME   (GMT)   \t=   "   +
datetime);
                    Console.WriteLine("              BUSY     \t\t=     "     +
snapshot.GetChildContent("busy"));
                    Console.WriteLine("       TOTAL   (of   1024B)   \t=   "   +
snapshot.GetChildContent("total"));
                    Console.WriteLine("    CUMULATIVE TOTAL (of 1024B) = " +
snapshot.GetChildContent("cumulative-total"));
                    Console.WriteLine("           DEPENDENCY    \t\t=    "    +
snapshot.GetChildContent("dependency"));
                }
            }
            catch (NaAuthException e)
            {
                Console.Error.WriteLine("Authentication    Error    :    "    +
e.Message);
            }
            catch (NaApiFailedException e)
            {
                Console.Error.WriteLine("API Failed : " + e.Message );
            }

        }


        static void GetSchedule(String[] args)
        {
            NaServer s;
            string filer = args[1];
            string user = args[2];
            string pwd = args[3];
            string vol = args[4];

            NaElement xi, xo;
            //
            // get the schedule
            //
            try
            {
                s = new NaServer(filer, 1, 0);
                s.Style = NaServer.AUTH_STYLE.LOGIN_PASSWORD;
                s.TransportType = NaServer.TRANSPORT_TYPE.HTTP;
                s.SetAdminUser(user, pwd);

                xi = new NaElement("snapshot-get-schedule");
                if (args.Length == 5)
                {
                    xi.AddNewChild("volume", vol);
                }
                else
```

```
                {
                    Console.Error.WriteLine("Invalid number of arguments");
                    Usage(args);
                    System.Environment.Exit(-1);
                }

                xo = s.InvokeElem(xi);
                //
                // print it out
                //
                Console.WriteLine("Snapshot schedule for volume "+ vol +   "
on filer " + filer + ":");
                Console.WriteLine("----------------------------------------
----------------------");
                Console.WriteLine("Snapshots are taken on minutes [" +
                            xo.GetChildIntValue("which-minutes",  0)  +  "]  of
each hour (" +
                            xo.GetChildContent("minutes") + " kept)" );
                Console.WriteLine("Snapshots are taken on hours [" +
                            xo.GetChildContent("which-hours")  +  "]  of  each
day (" +
                            xo.GetChildContent("hours") + " kept)\n");
                Console.WriteLine(xo.GetChildContent("days")   +   "   nightly
snapshots are kept\n");
                Console.WriteLine(xo.GetChildContent("weeks")   +   "   weekly
snapshots are kept\n");
                Console.WriteLine("\n");
            }
            catch (NaException e)
            {
                Console.WriteLine("ERROR: " + e.Message);
            }

        }
        static void Main(string[] args)
        {
            if (args.Length < 5)
                    Usage(args);

            Console.WriteLine("|-------------------------------------|");
            Console.WriteLine("| Program to Demo use of Snapshot APIs |");
            Console.WriteLine("|-------------------------------------|");

            switch (args[0]) {
                case "-g":
                    GetSchedule(args);
                    break;
                case "-c":
                    CreateSnapshot(args);
                    break;
                case "-r":
                    RenameSnapshot(args);
                    break;
                case "-d":
                    DeleteSnapshot(args);
                    break;
                case"-l":
```

```
                    ListInfo(args);
                    break;
                default:
                    Usage(args);
                    break;
            }

        }
    }
}
```