.NET Example

SystemAPI .NET Tool is a program written in C# allowing developers to manage VoipNow accounts and view information about calls.

- How To Install It
 - System Requirements
 - Setup
- How To Use It
 - Authentication
 Examples
 - amples
 - Add Related AccountsAdd a Service Provider Account
 - Add a Service Provider Account
 Add an Organization Account
 - Get the Call Costs for a User Account

How To Install It

System Requirements

In order to be able to use the SystemAPI .NET Tool, you need a system with:

- Microsoft Visual Studio 2010
- WSCF.blue (Web Services Contract First) add-in for Microsoft Visual Studio 2010. Download the version 1.0.10, extract the archive, and run the msi installer.

Setup

STEP 1: Please download the files from our GitHub repository:

git clone https://github.com/4psa/systemapi-example-net.git

STEP 2: Start Visual Studio 2010 and create a new Visual C# Console Application project.

STEP 3: Create a directory within your project and place the VoipNow schemes in it as you can see below.

You can find the schemas in your VoipNow installation server at this location:

/usr/local/voipnow/admin/htdocs/soap2/schema/latest/

Solution Explorer	₹₽X
Solution 'DotNET SOAP API 3.0.0' (1 project)	
DotNET SOAP API 3.0.0	
Properties	
References	
Service References	
3.0.0	
a 🗁 Account	
WSOL Account.wsdl	
孟 AccountData.xsd	
器 AccountMessages.xsd	
Billing	
wsot Billing.wsdl	
温 BillingData.xsd	
遇 BillingMessages.xsd	
A Channel	
wsot Channel.wsdl	
品 ChannelData.xsd	
品 ChannelMessages.xsd	
Extension	
wsou Extension.wsdl	
温 ExtensionData.xsd	
La ExtensionMessages.xsd	
a 🗁 GlobalOp	
wsot GlobalOp.wsdl	
员 GlobalOpData.xsd	
品 GlobalOpMessages.xsd	
Image: A constraint of the second	
wsoL Organization.wsdl	
B OrganizationData.xsd	
OrganizationMessages.xsd	
A Dep PBX	
wsoL PBX.wsdl	
BXData.xsd	
B PBXMessages.xsd	
A 🗁 Report	
wsou Report.wsdl	
函 ReportData.xsd	
A ReportMessages.xsd	
ServiceProvider	
ServiceProvider.wsdl	
ServiceProviderData.xsd	
ServiceProviderMessages.xsd	
⊿ User	
Vser.wsdl	
📇 UserData.xsd	
函 UserMessages.xsd	
a common.xsd	
A HeaderData.xsd	
wow voipnowservice.wsdl	



STEP 4: Right-click on the voipnowservice.wsdl file and then pick WSCF.blue Generate Web Service Code, using the following option:

CF.blue Code Generation	1.0		
			< wscf :
Contract information			
WSDL location: voipnows	ervice.wsdl		•
Code generation			
Olient-side proxy	O Service-side stub		
Options			
Public properties	Format SOAP Actions	Collections	List <t></t>
🗹 Data binding	Order identifiers	Async methods	Separate files
Adjust casing			
Service Behavior			
Concurrency Mode	Single	Use Synchron	ization Context
Instance Context Mode	PerCall	Enable WSDL	. Endpoint
		Generate .svo	file
Service Method Implement Generate a regular se Generate a partial ser Generate an abstract	ation rvice class with methods that th vice class with calls to partial m service class and abstract met	nrow a NotImplementedEx nethods that must be imple hods that can be implemen	ception in their body. mented in another file. nted in a derived class.
files and namespaces			
Destination file name	voip.cs		
b ootan dation mice manne			

In the 'Code generation' fieldset, Client-side proxy option is selected, since we generate the code for the client.

In the 'Options' fieldset, Public properties and Data Binding are checked.

- When Public properties is checked, the code generator will create property getters and setters for the data fields.
- When **Data Binding** is checked, the resulting types will automatically include the implementation of INotifyPropertyChanged interface, which is used to notify clients that a property value has changed.

In our examples, the destination namespace is set to voipnowsoap_net. Please note that if you change this value, you will have to replace the using voipnowsoap_net; line at the beginning of each application file to using <YOUR_NAMESPACE>; .

The WSCF tool will generate the proxy classes for you. In our example, the proxy classes have been created within the voip.cs file.

You_can now use the recently created classes to write applications that interact with your VoipNow SystemAPI.

How To Use It

The SystemAPI .NET Tool contains tests that simulates the following operations:

- add a Service Provider Account,
- add an Organization Account,
- add a User Account,
- add an Extension Account,
- get the Call Costs for a User Account.

You may find the tests under the path: <code><PATH_TO_SYSTEMAPI_TOOL>\Operations\Tests.cs</code> .

Also, you may find the implementation of the operations mentioned above under the path: <PATH_TO_SYSTEMAPI_TOOL>\Operations.

The entry point of the SystemAPI.NET Tool is the Main method from the Program class (<PATH_TO_SYSTEMAPI_TOOL>\Program.cs file). Here you may switch between running different tests.

The entry point of the SystemAPI .NET Tool

```
class Program
{
    static void Main(string[] args)
        // Change this with the oauth token of the account you want to use.
       string accessToken = "CHANGEME";
       Tests.AddRelatedAccounts(accessToken);
        // Uncomment the following line if you want to run this test
        //Tests.AddServiceProviderAccount(accessToken);
        // Uncomment the following line if you want to run this test
        //Tests.AddOrganizationAccount(accessToken);
        // Uncomment the following line if you want to run this test
        //Tests.AddUserAccount(accessToken);
        // Uncomment the following line if you want to run this test
        //Tests.AddExtensionAccount(accessToken);
    }
}
```

Authentication

Before running the tests you have to specify the **oauth token** of the account you want to use. In order to achieve this you will update the **accessToken** variable of the **Main** method from the **Program** class (Program.cs file) with a valid **oauth token**.

The place where you should set the oauth token

```
class Program
{
   static void Main(string[] args)
   {
      // Change this with the oauth token of the account you want to use.
      string accessToken = "CHANGEME";
      // Tests
      Tests.AddRelatedAccounts(accessToken);
      // Other tests...
   }
}
```

The outh token is used later when a userCredentials object is created before calling any method of the VoipNow SystemAPI Client-side Proxy.

```
Using the oauth token to create a userCredetials object
//...
userCredentials credentials = new userCredentials() { accessToken = accessToken };
//...
ServiceProviderClient serviceProviderClient = new ServiceProviderClient("ServiceProviderPort");
//...
serviceProviderClient.AddServiceProvider(credentials, request, out response);
```

All the test use the 4PSA demo server located at https://voipnow3demo.4psa.com/.

Examples

Add Related Accounts

The below code snippet shows how to add related accounts starting with a Service Provider Account, and then adding an Organization Account, and then a User Account and finally an Extension Account is added.

It is important to note that after an account is added, the permissions and limits for the account are set. Also, each account adding operation outputs the added account **Id** which is used to add the next account which is a child of the currently added account. For example, the adding of a Service Provider Account outputs the Service Provider Account Id, which is being used to add an Organization Account.

AddRelatedAccounts test

```
public static void AddRelatedAccounts(string accessToken)
{
    // Step 1: add a service provider
   var serviceProviderId = ServiceProviderOperations.AddServiceProviderAccount(accessToken);
    11
    // Step 2: set the service provider permissions and limits
   ServiceProviderOperations.SetServiceProviderAccountPermissionAndLimits(accessToken, serviceProviderId);
    11
    // Step 3: add an organization to the service provider created at the Step 1.
   var organizationID = OrganizationOperations.AddOrganizationAccount(accessToken, serviceProviderId);
    11
    // Step 4: set the organization permissions and limits
   OrganizationOperations.SetOrganizationAccountPermissionsAndLimits(accessToken, organizationID);
    11
   // Step 5: add an user to the organization created at the Step 3.
   var userID = UserOperations.AddUserAccount(accessToken, organizationID);
    // Step 6: set the user permissions and limits
   UserOperations.SetUserAccountPermissionsAndLimits(accessToken, userID);
    11
    // Step 7: add an extension to the user created at the Step 5.
   ExtensionOperations.AddExtensionAccount(accessToken, userID);
   Console.Read();
}
```

In order to run the above test, the oauth token must be provided. The test can be found in the file: <PATH_TO_SYSTEMAPI_TOOL>\Operations\Tests .cs.

Add a Service Provider Account

The below code snippet shows how to add a Service Provider Account. The test can be found inside the file <PATH_TO_SYSTEMAPI_TOOL>\Operations\Tests.cs.

```
Add a Service Provider Account

public static void AddServiceProviderAccount(string accessToken)
{
    ServiceProviderOperations.AddServiceProviderAccount(accessToken);
    Console.Read();
}
```

In order to run this test, the oauth token must be provided.

The next code snippet shows how the Client-side Proxy is used in order to call a VoipNow SystemAPI operation for adding a Service Provider Account.

```
Calling the VoipNow SystemAPI for adding a Service Provider Account
       public static string AddServiceProviderAccount(string accessToken)
        {
            if (string.IsNullOrEmpty(accessToken))
            {
                Console.WriteLine("The access token cannot be null or empty!");
                return null;
            ServicePointManager.ServerCertificateValidationCallback += delegate(object sender,
                X509Certificate certificate, X509Chain chain, SslPolicyErrors sslPolicyErrors)
            {
                return true;
           };
            ServicePointManager.Expect100Continue = false;
           userCredentials credentials = new userCredentials() { accessToken = accessToken };
           AddServiceProvider request = CreateAddServiceProviderRequest(new Random());
            AddServiceProviderResponse response = new AddServiceProviderResponse();
            ServiceProviderClient serviceProviderClient = new ServiceProviderClient("ServiceProviderPort");
           Console.WriteLine("Adding a Service Provider...");
            try
            {
                serviceProviderClient.AddServiceProvider(credentials, request, out response);
            catch (Exception e)
            ł
                //exception found, so we check the stack trc
                String trace = e.StackTrace;
                //write the stack trace to the console
                Console.WriteLine("{0} Exception caught.", e);
                //wait for the user to press a key before closing the console
                Console.Read();
            }
           finally
            {
                Console.WriteLine("The response for adding a Service Provider:");
                foreach (PropertyDescriptor descriptor in TypeDescriptor.GetProperties(response))
                {
                    string name = descriptor.Name;
                    object value = descriptor.GetValue(response);
                    Console.WriteLine("\t{0}={1}", name, value);
                }
                Console.WriteLine();
            }
           return response.ID;
        }
```

Add an Organization Account

The below code snippet shows how to add an Organization Account. The test can be found inside the file <PATH_TO_SYSTEMAPI_TOOL>\Operations \Tests.cs.

```
Add an Organization Account
public static void AddOrganizationAccount(string accessToken)
{
    string serviceProviderId = "CHANGEME";
    OrganizationOperations.AddOrganizationAccount(accessToken, serviceProviderId);
    Console.Read();
}
```

In order to run this test, the **oauth token** must be provided (i.e. accessToken parameter). Also, you must change the value of the serviceProviderId variable from "CHANGEME" to a valid Service Provider Account Id.

The next code snippet shows how the Client-side Proxy is used in order to call a VoipNow SystemAPI operation for adding an Organization Account.

Calling the VoipNow SystemApi for adding an Organization Account

```
public static string AddOrganizationAccount(string accessToken, string parentServiceProviderID)
       {
           if (string.IsNullOrEmpty(parentServiceProviderID))
           {
               Console.WriteLine("The parentServiceProviderID parameter cannot be null or empty");
               return null;
           if (string.IsNullOrEmpty(accessToken))
           {
               Console.WriteLine("The access token cannot be null or empty!");
               return null;
           }
           ServicePointManager.ServerCertificateValidationCallback += delegate(object sender,
               X509Certificate certificate, X509Chain chain, SslPolicyErrors sslPolicyErrors)
           {
               return true;
           };
           ServicePointManager.Expect100Continue = false;
           userCredentials credentials = new userCredentials() { accessToken = accessToken };
           AddOrganization request = CreateOrganizationRequest(new Random(), parentServiceProviderID);
           AddOrganizationResponse response = new AddOrganizationResponse();
           OrganizationClient organizationClient = new OrganizationClient("OrganizationPort");
           Console.WriteLine("Adding an Organization");
           try
           {
               organizationClient.AddOrganization(credentials, request, out response);
           catch (Exception e)
           {
               //exception found, so we check the stack trc
               String trace = e.StackTrace;
               //write the stack trace to the console
               Console.WriteLine("{0} Exception caught.", e);
               //wait for the user to press a key before closing the console
               Console.Read();
           }
           finally
           {
               Console.WriteLine("The operation response:");
               foreach (PropertyDescriptor descriptor in TypeDescriptor.GetProperties(response))
               {
                   string name = descriptor.Name;
                   object value = descriptor.GetValue(response);
                   Console.WriteLine("\t{0}={1}", name, value);
               }
               Console.WriteLine();
           return response.ID;
       }
```

Get the Call Costs for a User Account

The below code snippet shows how to get the call costs for a **User Account**. The test can be found inside the file <PATH_TO_SYSTEMAPI_TOOL>\Opera tions\Tests.cs.

Get the Call Costs for a User Account

```
public static void GetCallCosts(string accessToken)
{
    string userID = "CHANGEME";
    CallCostsOperations.GetCallCosts(accessToken, userID);
    Console.Read();
}
```

In order to run this test, the **oauth token** must be provided (i.e. accessToken parameter). Also, you must change the value of the userId variable from "CHANGEME" to a valid User Account Id.

The next code snippet shows how the Client-side Proxy is used in order to call a VoipNow SystemAPI operation for getting the call costs for a **User** Account.

```
Calling the VoipNow SystemApi for getting the call costs for a User Account
public static void GetCallCosts(string accessToken, string userID)
        {
            if (string.IsNullOrEmpty(userID))
            {
                Console.WriteLine("The userID parameter cannot be null or empty");
                return;
            }
            if (string.IsNullOrEmpty(accessToken))
            {
                Console.WriteLine("The access token cannot be null or empty!");
                return;
            }
            ServicePointManager.ServerCertificateValidationCallback += delegate(object sender,
                X509Certificate certificate, X509Chain chain, SslPolicyErrors sslPolicyErrors)
            {
                return true;
            };
            ServicePointManager.Expect100Continue = false;
            userCredentials credentials = new userCredentials() { accessToken = accessToken };
            CallCosts request = CreateCallCostsRequest(userID);
            CallCostsResponse response = new CallCostsResponse();
            ReportClient reportClient = new ReportClient("ReportPort");
            Console.WriteLine("Getting the call costs for the user with the id = \{0\}", userID);
            try
            {
                reportClient.CallCosts(credentials, request, out response);
            catch (Exception e)
            {
                //exception found, so we check the stack trc
                String trace = e.StackTrace;
                //write the stack trace to the console
                Console.WriteLine("{0} Exception caught.", e);
                //wait for the user to press a key before closing the console
                Console.Read();
            }
            finally
            {
                Console.WriteLine("The operation response:");
                foreach (PropertyDescriptor descriptor in TypeDescriptor.GetProperties(response))
                {
                    string name = descriptor.Name;
                    object value = descriptor.GetValue(response);
                    Console.WriteLine("\t{0}={1}", name, value);
                Console.WriteLine();
            }
        }
```