
Parallels

Parallels Agent

SOAP API Tutorial

1.0



(c) 1999-2008

ISBN: N/A
SWsoft
13755 Sunrise Valley Drive
Suite 325
Herndon, VA 20171
USA
Tel: +1 (703) 815 5670
Fax: +1 (703) 815 5675

© 1999-2007 SWsoft. All rights reserved.
Distribution of this work or derivative of this work in any form is prohibited unless prior written permission is obtained from the copyright holder.
Virtuozzo, Plesk, HSPcomplete, and corresponding logos are trademarks of SWsoft.
Virtuozzo is a patented virtualization technology protected by U.S. patents 7,099,948; 7,076,633; 6,961,868.
Patents pending in the U.S.
Plesk and HSPcomplete are patented hosting technologies protected by U.S. patents 7,099,948; 7,076,633.
Patents pending in the U.S.
Intel, Pentium, and Celeron are registered trademarks of Intel Corporation.
IBM DB2 is a registered trademark of International Business Machines Corp.
MegaRAID is a registered trademark of American Megatrends, Inc.
PowerEdge is a trademark of Dell Computer Corporation.

Contents

Preface	5
Documentation Conventions.....	5
Typographical Conventions.....	5
Shell Prompts in Command Examples	6
General Conventions	6
Feedback	6
Introduction	7
What is Parallels Agent?.....	7
Agent SOAP API.....	8
Development Platforms	8
Installation	9
Choosing Development Project	10
Generating Proxy Classes From WSDL	11
Errors and Resolution	12
Creating a Simple Client Program	13
Main Program File.....	14
Certificates Policy Preparation	14
Instantiating Proxy Classes	15
Connection URL.....	18
Logging in and Creating a Session	18
Retrieving a List of Virtuozzo Containers	20
Complete Program Code.....	22
Virtuozzo Container Management	27
Creating a Virtuozzo Container	28
Getting Server ID From Name.....	32
Starting, Stopping, Restarting a Container.....	33
Destroying a Container	35
Suspending and Resuming a Container	36
Getting Container Configuration Information	37
Configuring a Container	38
Modifying IP Address	38
Modifying Hostname.....	41
Modifying Container Name.....	42
Modifying QoS Settings	43
Modifying DNS Server Assignment.....	44

Cloning a Virtuozzo Container	45
Migrating a Container to a Different Host	47
Performance Monitor	50
Classes, Instances, Counters	51
Getting a Performance Report	54
Monitoring Alerts	58
File Management	60
Request Routing	61
Listing Files	63
Uploading a File	65
Downloading a File	67
Index	68

CHAPTER 1

Preface

In This Chapter

Documentation Conventions.....	5
Feedback	6

Documentation Conventions

Before you start using this guide, it is important to understand the documentation conventions used in it. For information on specialized terms used in the documentation, see the Glossary at the end of this document.

Typographical Conventions

The following kinds of formatting in the text identify special information.

Formatting convention	Type of Information	Example
Triangular Bullet(➤)	Step-by-step procedures. You can follow the instructions below to complete a specific task.	<i>To create a VE:</i>
Special Bold	Items you must select, such as menu options, command buttons, or items in a list.	Go to the Resources tab.
	Titles of chapters, sections, and subsections.	Read the Basic Administration chapter.
<i>Italics</i>	Used to emphasize the importance of a point, to introduce a term or to designate a command line placeholder, which is to be replaced with a real name or value.	These are the so-called <i>EZ templates</i> . To destroy a VE, type <code>vzctl destroy veid</code> .
Monospace	The names of commands, files, and directories.	Use <code>vzctl start</code> to start a VE.
Preformatted	On-screen computer output in your command-line sessions; source code in XML, C++, or other programming languages.	Saved parameters for VE 101
Monospace Bold	What you type, contrasted with on-screen computer output.	# rpm -V virtuoizzo-release
CAPITALS	Names of keys on the keyboard.	SHIFT, CTRL, ALT

KEY+KEY	Key combinations for which the user must press and hold down one key and then press another.	CTRL+P, ALT+F4
---------	--	----------------

Shell Prompts in Command Examples

Command line examples throughout this guide presume that you are using the Bourne-again shell (bash). Whenever a command can be run as a regular user, we will display it with a dollar sign prompt. When a command is meant to be run as root, we will display it with a hash mark prompt:

Bourne-again shell prompt	\$
Bourne-again shell root prompt	#

General Conventions

Be aware of the following conventions used in this book.

- Chapters in this guide are divided into sections, which, in turn, are subdivided into subsections. For example, **Documentation Conventions** is a section, and **General Conventions** is a subsection.
- When following steps or using examples, be sure to type double-quotes ("), left single-quotes ('), and right single-quotes (') exactly as shown.
- The key referred to as RETURN is labeled ENTER on some keyboards.

The root path usually includes the `/bin`, `/sbin`, `/usr/bin` and `/usr/sbin` directories, so the steps in this book show the commands in these directories without absolute path names. Steps that use commands in other, less common, directories show the absolute paths in the examples.

Feedback

If you spot a typo in this guide, or if you have thought of a way to make this guide better, we would love to hear from you!

If you have a suggestion for improving the documentation (or any other relevant comments), try to be as specific as possible when formulating it. If you have found an error, please include the chapter/section/subsection name and some of the surrounding text so we can find it easily.

Please submit a report by e-mail to userdocs@swsoft.com.

Introduction

What is Parallels Agent?

Parallels Agent is a server-side software that allows client applications to connect to and manage Virtuozzo Containers over network. Agent can be used for managing, monitoring, and tuning the Hardware Node and Virtuozzo Containers.

The following list describes the most common tasks that can be performed on Virtuozzo Containers through Agent:

- Creating and destroying a Container.
- Starting, stopping, restarting.
- Migrating, cloning, moving to a different location.
- Backing up.
- Getting the status and configuration information.
- Modifying configuration parameters.
- Obtaining current statistical data and resource usage information.
- Setting up Virtuozzo Virtual Networks.
- Managing Virtuozzo Security Infrastructure.
- Installing, updating, removing Virtuozzo templates.

The following tasks can be performed on Hardware Nodes and Virtuozzo Containers:

- Shutting down and restarting.
- Managing configuration parameters.
- Managing operating system services.
- Managing devices.
- Managing files and directories.
- Managing users and groups.
- Retrieving disk, network and other system information.
- Monitoring resource consumption.
- Receiving notifications about critical events, directly or via e-mail.

Agent SOAP API

Agent SOAP API is based on open standards like SOAP and WSDL. With SOAP API, you build your client applications using one of the third-party development tools that can generate client code from WSDL specifications. The code generated from WSDL documents is a set of objects in your application's native programming language. You work with data structures using object properties and you make API calls by invoking object methods.

The SOAP API shares the XML schema with the Agent XML API, so the basic format of the input and output data is the same in both APIs. *Parallels Agent Programmer's Guide, Using XML API* chapter provides general information on the Agent XML schema, the detailed description of the XML API request and response packets, and other important information. *Parallels Agent XML Programmer's Reference* provides a complete XML API reference. When working with SOAP API, use the XML API reference material to find the descriptions of the calls, their input and output parameters, and XML code examples.

Development Platforms

In this tutorial we will write our sample code in C# using Microsoft Visual Studio .NET 2005 and Microsoft .NET Framework 2.0.

Agent SOAP API has also been successfully tested with Microsoft Visual Studio .NET 2003 and Microsoft .NET Framework 1.1

Installation

Server side

Agent software is included in the Virtuozzo Tools package which comes with Parallels Virtuozzo Containers. Virtuozzo Tools are installed on your server by default during Virtuozzo Containers installation. If Virtuozzo Tools are not installed on your server, run Virtuozzo Containers installation program again and select Virtuozzo Tools package to install.

When Agent is installed on your host server for the first time, you will need to know the password of your system administrator (such as `root` on Linux or `Administrator` on Windows) in order to log in to it from your client program. The system administrator is by default granted all access rights in Agent, which means that the user can execute any of the available Agent API calls and access any of the Virtuozzo Containers hosted by the Hardware Node. You can add more users with specific access rights later using Virtuozzo Tools or programmatically through Agent.

To verify that Agent is installed and running properly, do the following:

 On Linux, log in to your Hardware Node and execute the following command:

```
# vzagent_ctl status
```

If Agent is running, the output should look similar to the following:

```
# vzagent (pid 31615 29644 25012 22861 8362 7073 7046 7036 7035 7029
7028 7026 7025 7023 7021 7019 7018 7017 7016 7013 7012 7011 7010 7009
7008 7007 7006 7004 7003 7002 7001 7000 6999 6998 6997 6996 6995 6994
6993 6992 6991 6990 6989 6988 6987 6986 6985 6984 6632) is running...
```

If Agent is stopped, the output will look like this:


```
# vzagent is stopped
```

If something is wrong with Agent, the output may contain additional messages describing the problem. In such a case, try restarting Agent using the following command:

```
# vzagent_ctl restart
```

To start or stop Agent, use the following commands respectively:

```
# vzagent_ctl start
# vzagent_ctl stop
```

 On Windows, Agent runs as a Windows service. You can manipulate it by going to the Services console which is located in the Control Panel / Administrative Tools folder, and selecting the `VZAgent` service from the list.

Client side

You will need Microsoft Visual Studio .NET and Microsoft .NET Framework installed on your development machine. No additional client software is required.

Choosing Development Project

You can choose any type of Visual Studio .NET C# project for your application. Your choice depends on your application requirements only. For our sample program, let's select C# Windows console application project and call it `VzSimpleClient`.

- 1 In Microsoft Visual Studio .NET, select **File > New > Project**. The **New Project** windows opens.
- 2 In the **Project Types** tree, select **Visual C# > Windows** and then select **Console Application** in the **Templates** pane.
- 3 Enter `VzSimpleClient` as the name for your project and choose a location for your project files and click **OK**.

Note: If you are using Microsoft Visual Studio .NET 2005 and if your default project files location is set to `C:\Documents and Settings\user_name\My Documents\Visual Studio 2005\Projects\project_name\..`, you will have to choose a location with a shorter path. The reason is that there's an issue with Visual Studio 2005 C# method generation from WSDL (we will discuss the issue in detail in the **Generating Stubs From WSDL** section). As a solution, we will create a batch file that will fix the problem. The file will be placed into and run from the directory that contains the **Web References** folder (usually `..\Projects\project_name\project_name\`), but because of the 256 character command line limit imposed by the Microsoft NTFS file system, the full pathname (including the path and the file name) must fit within this limit or the C# compiler will not be able to run the batch file.

Generating Proxy Classes From WSDL

- 1 In the **Solution Explorer** pane, select the `VzSimpleClient` project.
- 2 On the **Project** menu, select **Add Web Reference**. The **Add Web Reference** window opens.

In the **URL** field, type (or copy and paste) this URL:
`http://www.swsoft.com/webservices/vza/4.0.0/VZA.wsdl`

- 1 Press the **Go** button next to the **URL** field. Visual Studio will try to connect to the SWsoft web site and retrieve the Agent web service information. After a few seconds (depends on the connection speed), you should see a single entry in the **Web services found at this URL** list box: 1 Service Found: - VZA
- 2 Type `VZA` in the **Web reference name** field replacing the default value (in general, you can choose any name that you like). This name will be used in your code as the C# namespace to access the selected service.

Press the **Add Reference** button. This will generate proxy classes from Agent WSDL specifications and will add them to the project. A new item `VZA` will appear in the **Solution Explorer** in the **Web References** folder. You can now start using generated classes to access Agent services.

Errors and Resolution

If you are using Microsoft Visual Studio 2005, you may get errors when generating client code from WSDL. The errors may look similar to the following:

```
error CS0542: 'set_xxx': member names cannot be the same as their
enclosing type
```

The error is produced by the C# compiler when generating the code for the XML schema similar to the following example:

```
<xs:element name="set_xxx">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="xxx" type="XXXtype" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Note that the function `set_xxx` has a parameter `xxx`. Microsoft Visual C# .NET will generate the following code:

```
public partial class set_xxx {
    private string xxxField;

    public string xxx {
        get {
            return this.xxxField;
        }
        set {
            this.xxxField = value;
        }
    }
}
```

As you can see, the function name is the same as the class name. This causes the compiler to produce those errors.

Resolution:

Create a batch file `wsdlc.bat` containing the following code and save it in your project directory:

```
setlocal
set WS=%1Web References\VZA
copy "%WS%\Reference.map" "%WS%\Reference.discomap"
"C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\wsdl.exe"
/l:CS /fields /out:"%WS%\Reference.cs" /n:%2.VZA
"%WS%\Reference.discomap"
del "%WS%\Reference.discomap"
endlocal
exit /b 0
```

The file generates the new `Reference.cs` file (the file containing the proxy classes) fixing the problem described above by generating the regular properties instead of C#-style get/set fields. DO NOT try to run the file! It will be run automatically after we complete the rest of the steps.

In the Microsoft Visual C# .NET development environment, select **Project > Properties** menu item. Select **Build Events** option in the left pane. Now in the right pane, modify the parameter **Pre-build Event Command Line** to contain the following line:

```
$(ProjectDir)wsdlc.bat $(ProjectDir) $(ProjectName)
```

Note: Make sure that the `Reference.cs` file is not currently opened in the IDE, otherwise the compiler will use it instead of the new file that will be generated by our batch file.

Select the **Build > Build Solution** menu option to build your solution. This will take longer than usual because the `wsdlc.bat` file that we created will re-generate the proxy classes.

After the build is completed, the `Reference.cs` file will contain the newly generated stubs. At this point you can remove or comment out the entry that used in the **Project > Properties > Pre-build Event Command Line** option. If you do not, the stubs will be re-generated every time you build your solution.

If you decide to update the client code from WSDL located on our Web server again, make sure that you repeat the steps described here again.

Creating a Simple Client Program

The request describing this defect was submitted to Microsoft: #FDBK46565

In this section, we will create a simple Agent client application that will log the specified user in and will retrieve the list of the Virtuozzo Containers from the Hardware Node. The complete program code is included in the Complete Program Code section.

Main Program File

At this point, you should see the `Program.cs` file opened in your Visual Studio IDE. This is the main file where we will write our program code. The file should contain the following code:

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;
using VzSimpleClient.VZA;

namespace VzSimpleClient
{
    class Program
    {
        static void Main(string[] args)
        {
            // Wait for the user to press a key, then exit.
            Console.Read()
        }
    }
}
```

We've added the necessary using directives and we've also added the `Console.Read()` line to the `Main()` function to keep the console window open until a keyboard key is pressed.

Certificates Policy Preparation

Since Agent SOAP uses HTTPS as a transport protocol, we have to deal with the certificate issues. For the purpose of this example, we're going to use the "trust all certificates" policy. We'll create a class that implements such a policy for us and passes it to the certificate policy manager during logon.

```
///<summary>
/// Sample class TrustAllCertificatePolicy.
/// Used as a certificate policy provider.
/// Allows all certificates.
///</summary>
public class TrustAllCertificatePolicy : System.Net.ICertificatePolicy
{
    public TrustAllCertificatePolicy()
    { }

    public bool CheckValidationResult(System.Net.ServicePoint sp,
        System.Security.Cryptography.X509Certificates.X509Certificate
cert,
        System.Net.WebRequest req, int problem)
    {
        return true;
    }
}
```

Instantiating Proxy Classes

When Visual Studio .NET generates proxy classes for Web services, it names them by taking a Web service name and appending the "Binding" string to it. For example, the name of the proxy class for invoking the `vzaenvm` service will be `vzaenvmBinding`; the `filer` service will have a proxy class named `filerBinding`, etc.

Instantiating a proxy class (creating an object from it) is not as straightforward as creating an ordinary object in your C# program. In this section, we will create a sample class that will provide methods for creating an object from a proxy class. In addition, the methods of the class will also set up and populate the header portion of the Agent request message that will be sent to Agent. Agent request message header contains parameters that provide information on how the request should be handled on the server side. The most important of those are:

Parameter Name	Description
<code>session</code>	Agent session ID. The session is established on the server side after successful login and the session ID is returned to the client program. Each subsequent Agent request sent from the client must include this ID in order to be recognized and approved by Agent.
<code>target</code>	<p>The name of the Agent operator to which this request should be sent for processing. Each Web service has a target operator. Both the Web service and the corresponding operator have the same name. As you already know from the beginning of this section, the first part of the proxy class name is the name of the Web service, so it is the name of the target operator.</p> <p>For example, when invoking the <code>vzaenvmBinding</code> object, this parameter should contain <code>vzaenvm</code>. When invoking the <code>filerBinding</code> object, the name of the target operator is <code>filer</code>, and so forth.</p> <p>Note: There's a single Web service that is an exception to this rule. The <code>system</code> service (proxy class: <code>systemBinding</code>) actually requires an omission of this parameter from the request. For all other services the parameter must be appropriately set.</p>
<code>dst/host</code>	This parameter is used to specify the Server ID of the Virtuozzo Container to which the request should be routed. The parameter should be used with some of the Web services and it should be ignored with the others. We will talk in detail about request routing and will provide examples later in the tutorial. The parameter will be ignored in the beginning sections of the tutorial.

```

Sample Class:
/// <summary>
/// Sample class Binder.
/// Provides methods to create the specified binding object
/// and to populate the Agent message header.
/// </summary>
public class Binder
{
    string URL; // Agent server URL.
    string session; // Agent session ID.

    // Constructor. Sets URL and session ID values.
    public Binder(string url, string sess)
    {
        URL = url;
        session = sess;
    }

    /// <summary>
    /// Method InitBinding (overloaded).
    /// Instantiates a proxy class.
    /// <param name="bindingType">
    /// The System.Type object for a proxy class.
    /// To obtain the object, use the typeof operator
    /// with the name of the proxy class as a parameter.
    /// </param>
    /// <returns>
    /// <para>New proxy class object.</para>
    /// </returns>
    /// </summary>
    public System.Object InitBinding(System.Type bindingType)
    {
        System.Object Binding =

name
is
requires
        bindingType.GetConstructor(System.Type.EmptyTypes).Invoke(null);

        // Set URL.
        bindingType.GetProperty("Url").SetValue(Binding, URL, null);

        // Create the request message header object.
        packet_headerType header = new packet_headerType();

        // Set session ID.
        header.session = session;

        /* Set the "target" parameter in the Agent request
         * message header. The parameter must contain the name
         * of the corresponding Agent operator.
         * The operator name can be obtained from the name of the
         * proxy class. It is the substring from the beginning of the
         * followed by the "Binding" substring. For example, the name
         * of the corresponding operator for the "filerBinding" class
         * "filer".
         * All Agent requests except "system" requests must have the
         * target operator value set. System is the only operator that
         * the omission of the "target" parameter from the header.
         */
    }
}

```



```

        if (bindingType != typeof(systemBinding)) {
            header.target = new string[1];
            header.target[0] = bindingType.Name.Replace("Binding",
""");
        }

        // Set the request message header.
        bindingType.GetField("packet_header").SetValue(Binding,
header);
        return Binding;
    }

    /// <summary>
    /// Method InitBinding (overloaded).
    /// Instantiates a proxy class.
    /// Allows to set destination Container.
    /// </summary>
    /// <param name="bindingType">
    /// The System.Type object for a proxy class.
    /// To obtain the object, use the typeof operator
    /// with the name of the proxy class as a parameter.
    /// </param>
    /// <param name="eid">
    /// The Server ID of the destination Container to which to route
    /// the request message for processing.
    /// </param>
    /// <returns>
    /// <para>New proxy class object.</para>
    /// </returns>
    /// </returns>
    public System.Object InitBinding(System.Type bindingType, string
eid)
    {
        System.Object Binding =

bindingType.GetConstructor(System.Type.EmptyTypes).Invoke(null);

        // Set URL.
        bindingType.GetProperty("Url").SetValue(Binding, URL, null);

        // Create the request message header object.
        packet_headerType header = new packet_headerType();

        // Set session ID.
        header.session = session;

        /* Set the "target" parameter in the Agent request
        * message header.
        */
        if (bindingType != typeof(systemBinding)) {
            header.target = new string[1];
            header.target[0] = bindingType.Name.Replace("Binding",
""");
        }

        // Set the destination Server ID.
        header.dst.host = eid;

        // Set the request message header.

```

```

        bindingType.GetField("packet_header").SetValue(Binding,
header);
        return Binding;
    }
}

```

Connection URL

The Agent server listens for the secure HTTPS requests on port 4646. The connection URL will look similar to the following example (substitute the IP address value with the address of your server):

```
https://192.168.0.218:4646
```

You may also communicate with Agent using HTTP. In this case, the port number is 8080 and the URL should look like this:

```
http://192.168.0.218:8080
```

The URL will be used as an input parameter during the login procedure described in the following step.

Logging in and Creating a Session

The following is an example of a function that logs the user in using the supplied connection and login parameters.

Sample function parameters:

Name	Description
url	Agent server URL. See the Connection URL section (on page 18).
name	User name. In this tutorial, we will be login in as a system administrator of the host server (Hardware Node). You will need to know the password of your Hardware Node administrator account.
domain	We are not going to use this parameter in the tutorial. For more information on its usage, see Parallels Agent XML Programmer's Reference Guide .
realm	Realm ID. Realm is a database containing the user authentication information. Agent supports various types of authentication databases, including operating system user registries and LDAP-compliant directories, such as AD/ADAM on Windows and OpenLDAP on Linux. In our example, we will be using the user registry of the Hardware Node, which is called <i>System Realm</i> in Agent terminology. The unique ID that Agent uses for the System Realm is 00000000-0000-0000-0000-000000000000.

The function authenticates the specified user and, if the supplied credentials are valid, creates a session for the user and returns the session ID. All subsequent Agent requests must include the session ID in order to be recognized and approved by Agent. The Binder class will take care of including the session ID in the request message header.

Sample function:

```

/// <summary>
/// Sample function Login.
/// Authenticates the user using the specified credentials and
/// creates a new session.
/// </summary>
/// <param name="url">Agent server URL.</param>
/// <param name="name">User name.</param>
/// <param name="domain">Domain.</param>
/// <param name="realm">Realm ID.</param>
/// <param name="password">Password</param>
/// <returns>New session ID.</returns>
///
public string Login(string url, string name, string domain, string
realm, string password)
{
    try {
        System.Net.ServicePointManager.CertificatePolicy = new
TrustAllCertificatePolicy();

        // Login information object.
        login1 loginInfo = new login1();

        /* The sessionmBinding class provides the login and
        * session management functionality.
        */
        sessionmBinding sessionm = new VZA.sessionmBinding();

        /* Instantiate the System.Text.Encoding class that will
        * be used to convert strings to byte arrays.
        */
        System.Text.Encoding ascii = System.Text.Encoding.ASCII;

        // Populate the connection and the login parameters.
        sessionm.Url = url;
        loginInfo.name = ascii.GetBytes(name);
        if (domain.Length != 0) {
            loginInfo.domain = ascii.GetBytes(domain);
        }
        if (realm.Length != 0) {
            loginInfo.realm = realm;
        }
        loginInfo.password = ascii.GetBytes(password);

        // Log the specified user in.
        return sessionm.login(loginInfo).session_id;
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```

Retrieving a List of Virtuozzo Containers

The following function retrieves a list of Virtuozzo Containers from a Hardware Node. The function accepts a numeric code specifying the Container state as a parameter allowing you to retrieve the information only for the Containers in a particular state (running, stopped, etc.). The state codes are as follows:

Code	Name
0	Unknown
1	Unexisting
2	Config
3	Down
4	Mounted
5	Suspended
6	Running
7	Repairing
8	License Violation

The function returns a string containing the list of names of the existing Containers.

```

/// <summary>
/// sample function GetCTList.
/// Retrieves the list of Containers from a Hardware Node.
/// </summary>
/// <param name="state">Container state code.</param>
/// <returns>Container names.</returns>
///
public string GetCTList(int state)
{
    string list_result = "";

    try {
        // Instantiate the proxy class.
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));

        // The main input object.
        get_list1 velist = new get_list1();

        /* Set the Container status parameter.
        * -1 means ignore the status.
        */
        env_statusType[] env_status = new env_statusType[1];
        env_status[0] = new env_statusType();
        if (state == -1) {
            env_status[0].stateSpecified = false;
        }
        else {
            env_status[0].state = state;
        }
        velist.status = env_status;

        /* Get the list of the Containers then loop through it getting
the
        * Server ID and the name for each Container
        */
        foreach (string ve_eid in env.get_list(velist)) {
            get_info2 ve_info = new get_info2();
            ve_info.eid = new string[1];
            ve_info.eid[0] = ve_eid;

            /* Get the Container name from the Container configuration
structure.
            * Please note that if the name was not assigned to a
            * Container when it was created, the "name" field will be
empty.
            */
            list_result +=
env.get_info(ve_info)[0].virtual_config.name + "\n";
        }
    }
    catch (Exception e) {
        list_result += "Exception: " + e.Message;
    }
    return list_result;
}

```

Complete Program Code

```

using System;
using System.IO;
using System.Collections.Generic;
using System.Text;
using VzSimpleClient.VZA;

namespace VzSimpleClient
{
    class Program
    {
        Binder binder; // Binder object variable.
        string session_id = ""; // Agent session ID.

        // Main.
        static void Main(string[] args)
        {
            Program vzClient = new Program();
            try {
                vzClient.Run();
            }
            catch (System.Web.Services.Protocols.SoapException ex) {
                Console.WriteLine(ex.Code.ToString() + ", " +
ex.Message);
                Console.WriteLine("Details:" + ex.Detail.InnerText);
            }
            catch (System.Xml.XmlException xmlex) {
                Console.WriteLine(xmlex.ToString());
            }
            catch (System.InvalidOperationException opex) {
                Console.WriteLine(opex.Message + "\n" +
opex.InnerException);
            }
            Console.WriteLine("Press Enter to conintinue...");
            Console.Read();
        }

        ///<summary>
        /// Sample class TrustAllCertificatePolicy.
        /// Used as a certificate policy provider.
        /// Allows all certificates.
        ///</summary>
        public class TrustAllCertificatePolicy :
System.Net.ICertificatePolicy
        {
            public TrustAllCertificatePolicy()
            { }

            public bool CheckValidationResult(System.Net.ServicePoint
sp,
System.Security.Cryptography.X509Certificates.X509Certificate cert,
System.Net.WebRequest req, int problem)
            {
                return true;
            }
        }
    }
}

```

```

    }

    /// <summary>
    /// Sample class Binder.
    /// Provides methods to create the specified binding object
    /// and to populate the Agent message header.
    /// </summary>
    public class Binder
    {
        string URL; // Agent server URL.
        string session; // Agent session ID.

        // Constructor. Sets URL and session ID values.
        public Binder(string url, string sess)
        {
            URL = url;
            session = sess;
        }

        /// <summary>
        /// Method InitBinding (overloaded).
        /// Creates a binding object.
        /// <param name="bindingType">
        /// The name of the proxy class from which to
        /// create the object.
        /// </param>
        /// <returns>
        /// <para>New binding object.</para>
        /// </returns>
        /// </summary>
        public System.Object InitBinding(System.Type bindingType)
        {
            System.Object Binding =
                bindingType.GetConstructor(System.Type.EmptyTypes).Invoke(null);

            // Set URL.
            bindingType.GetProperty("Url").SetValue(Binding, URL,
null);

            // Create the request message header object.
            packet_headerType header = new packet_headerType();

            // Set session ID.
            header.session = session;

            /* Set the "target" parameter in the Agent request
             * message header. The parameter must contain the name
             * of the corresponding Agent operator.
             * The operator name can be obtained from the name of
the
             * proxy class. It is the substring from the beginning
of the name
             * followed by the "Binding" substring. For example,
the name
             * of the corresponding operator for the
"filerBinding" class is
             * "filer".
             * All Agent requests except "system" requests must
have the

```

```

        * target operator value set. System is the only
operator that requires
        * the omission of the "target" parameter from the
header.

        */
        if (bindingType != typeof(systemBinding)) {
            header.target = new string[1];
            header.target[0] =
bindingType.Name.Replace("Binding", "");
        }

        // Set the request message header.

bindingType.GetField("packet_header").SetValue(Binding, header);
        return Binding;
    }

    /// <summary>
    /// Method InitBinding (overloaded).
    /// Creates a binding object.
    /// Allows to set destination Container.
    /// </summary>
    /// <param name="bindingType">
    /// The name of the proxy class from which
    /// to create the object.
    /// </param>
    /// <param name="eid">
    /// The Server ID of the destination Container to which to
route
    /// the request message for processing.
    /// </param>
    /// <returns>
    /// <para>New binding object.</para>
    /// </returns>
    /// </returns>
    public System.Object InitBinding(System.Type bindingType,
string eid)
    {
        System.Object Binding =

bindingType.GetConstructor(System.Type.EmptyTypes).Invoke(null);

        // Set URL.
        bindingType.GetProperty("Url").SetValue(Binding, URL,
null);

        // Create the request message header object.
        packet_headerType header = new packet_headerType();

        // Set session ID.
        header.session = session;

        /* Set the "target" parameter in the Agent request
        * message header.
        */
        if (bindingType != typeof(systemBinding)) {
            header.target = new string[1];
            header.target[0] =
bindingType.Name.Replace("Binding", "");
        }
    }

```



```

        // Set the destination Server ID.
        header.dst.host = eid;

        // Set the request message header.

bindingType.GetField("packet_header").SetValue(Binding, header);
        return Binding;
    }
}

/// <summary>
/// Sample function Login.
/// Authenticates the user using the specified credentials and
/// creates a new session.
/// </summary>
/// <param name="url">Agent server URL.</param>
/// <param name="name">User name.</param>
/// <param name="domain">Domain.</param>
/// <param name="realm">Realm ID.</param>
/// <param name="password">Password</param>
/// <returns>New session ID.</returns>
///
public string Login(string url, string name, string domain,
string realm, string password)
{
    try {
        System.Net.ServicePointManager.CertificatePolicy = new
TrustAllCertificatePolicy();

        // Login information object.
        login1 loginInfo = new login1();

        /* The sessionmBinding class provides the login and
        * session management functionality.
        */
        sessionmBinding sessionm = new VZA.sessionmBinding();

        /* Instantiate the System.Text.Encoding class that
will
        * be used to convert strings to byte arrays.
        */
        System.Text.Encoding ascii =
System.Text.Encoding.ASCII;

        // Populate the connection and the login parameters.
        sessionm.Url = url;
        loginInfo.name = ascii.GetBytes(name);
        if (domain.Length != 0) {
            loginInfo.domain = ascii.GetBytes(domain);
        }
        if (realm.Length != 0) {
            loginInfo.realm = realm;
        }
        loginInfo.password = ascii.GetBytes(password);

        // Log the specified user in.
        return sessionm.login(loginInfo).session_id;
    }
    catch (Exception e) {

```

```

        return "Exception: " + e.Message;
    }
}

/// <summary>
/// sample function GetCTList.
/// Retrieves the list of Virtuozzo Containers from the
Hardware Node.
/// </summary>
/// <param name="state">Container state code.</param>
/// <returns>Container names.</returns>
///
public string GetCTList(int state)
{
    string list_result = "";

    try {
        // Instantiate the proxy class.
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));

        // The main input object.
        get_list1 velist = new get_list1();

        /* Set the Container status parameter.
        * -1 means ignore the status.
        */
        env_statusType[] env_status = new env_statusType[1];
        env_status[0] = new env_statusType();
        if (state == -1) {
            env_status[0].stateSpecified = false;
        }
        else {
            env_status[0].state = state;
        }
        velist.status = env_status;

        /* Get the list of the Containers then loop through it
getting the
        * Server ID and the name for each Container
        */
        foreach (string ve_eid in env.get_list(velist)) {
            get_info2 ve_info = new get_info2();
            ve_info.eid = new string[1];
            ve_info.eid[0] = ve_eid;

            /* Get the Container name from the Container
configuration structure.
            * Please note that if name was not assigned to a
            * Container when it was created, the "name" field
will be empty.
            */
            list_result +=
env.get_info(ve_info)[0].virtual_config.name + "\n";
        }
    }
    catch (Exception e) {
        list_result += "Exception: " + e.Message;
    }
    return list_result;
}

```

```

    }

    /// <summary>
    /// The Run() function is called from Main().
    /// It contains the code that executes other sample functions.
    /// </summary>
    ///
    public void Run()
    {
        /* The Agent server URL. Use the IP of
        * your own Hardware Node here.
        */
        string url = "http://10.30.67.54:8080/";

        // User name.
        string user = "root";

        // Domain name.
        string domain = "";

        /* Realm ID.
        * We are using the "system" realm here, so the
        * user will be authenticated against the
        * host operating system user registry.
        */
        string realm = "00000000-0000-0000-0000-000000000000";
        string password = "1q2w3e";

        // Log the user in.
        session_id = this.Login(url, user, domain, realm,
password);
        Console.WriteLine("Session ID: " + session_id);
        Console.WriteLine();

        // Create the Binder object.
        if (binder == null) {
            binder = new Binder(url, session_id);
        }

        // Get the list of Containers from the Hardware Node.
        Console.WriteLine(GetCTLList(-1));
        Console.WriteLine();
    }
}

```

Virtuozzo Container Management

```

}

```

The material in this section provides sample code and explains how to perform the most common Virtuozzo Container management tasks.

Creating a Virtuozzo Container

When creating a new Virtuozzo Container, the following configuration parameters are mandatory and must be selected every time:

- **Sample configuration name.** Virtuozzo Containers software comes with a set of sample configurations that are installed on the Hardware Node at the time the Virtuozzo Containers software is installed. XML API provides the `env_samplem/get_sample_conf` call to retrieve the list of the available configurations. In the example provided in this section, the C# equivalent of that call is the `env_samplemBinding.get_sample_conf()` method.
- **Operating System Template.** The list of the available templates can be retrieved using the `vzapkgm.get_list` XML API call. The C# equivalent is `vzapkgmBinding.get_list` call. For simplicity, we are not including this call in the example because Virtuozzo Containers for Windows currently comes with just one OS template, and Virtuozzo for Linux has one template for each supported Linux distribution. For example, the standard Red Hat Linux OS template name is `redhat-as3-minimal`.

The rest of the parameters that we use in this example are optional but are typically used when a new Container is created. The following sample shows how to create a Virtuozzo Container.

Sample Function Parameters:

Name	Description
<code>name</code>	The name that you would like to use for the Container.
<code>os_template</code>	The name of the OS template to use for the Container.
<code>platform</code>	Operating system type: <code>linux</code> or <code>windows</code> . This parameter will be used in our function to select a sample configuration for the Container. If the sample configuration is compatible with the specified platform, we will use it. In a real application, you would probably select the sample configuration in advance and would pass its name to the method that actually creates a Container. In this example, we automate this task while providing a demonstration of how to retrieve the list of the available sample configurations.
<code>architecture</code>	CPU architecture, e.g. <code>x86</code> , <code>ia64</code> . This parameter, together with the <code>platform</code> parameter (above) will also be used to determine the sample configuration compatibility with the specified CPU architecture.
<code>hostname</code>	The hostname that you would like to use for the Container.
<code>ip</code>	The IP address to assign to the Container.
<code>netmask</code>	Netmask.
<code>network</code>	Network interface ID: <code>venet0</code> for Linux; <code>venet1</code> for Windows. These are the standard host-routed Virtuozzo network interfaces. For other network configuration scenarios, please refer to Parallels Agent XML Programmer's Reference .

offline_management	Specifies whether to turn the Container Offline Management feature on or off.
--------------------	---

Sample Function:

```

/// <summary>
/// Sample function CreateCT.
/// Creates a new Virtuozzo Container.
/// </summary>
/// <param name="name">Container name.</param>
/// <param name="os_template">OS template name.</param>
/// <param name="platform">Operating system type: linux or
windows.</param>
/// <param name="architecture">CPU architecture (x86, ia64)</param>
/// <param name="hostname">Container hostname.</param>
/// <param name="ip">Container IP address.</param>
/// <param name="netmask">Netmask.</param>
/// <param name="network">Network interface ID.</param>
/// <param name="offline_management">
/// A flag specifyin whether to turn the "offline management"
/// feature on or off.
/// </param>
/// <returns>Server ID of the new Container.</returns>
public string CreateCT(string name, string os_template, string
platform, string architecture, string hostname, string ip, string
netmask, string network, bool offline_management)
{
    try {
        // Instantiate the proxy class.
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));

        // The main input object.
        create create_input = new create();

        // Container configuration information.
        venv_configType1 veconfig = new venv_configType1();

        /* Retrieve the list of sample configurations.
        * Select the first one that is compatible with the
        * specified platform (Linux, Windows) and CPU architecture.
        */
        env_samplemBinding env_sample =
(env_samplemBinding)binder.InitBinding(typeof(env_samplemBinding));
        get_sample_conf get_sample = new get_sample_conf();
        sample_confType[] samples =
env_sample.get_sample_conf(get_sample);

        if (samples != null) {
            foreach (sample_confType sample in samples) {
                if (sample.env_config.os != null) {
                    if (sample.env_config.os.platform == platform &&
sample.env_config.architecture == architecture) {
                        // Set sample configuration ID.
                        veconfig.base_sample_id = sample.id;
                        break;
                    }
                }
            }
        }

        // Set OS template.
        templateType osTemplate = new templateType();

```

```

    osTemplate.name = os_template;
    veconfig.os_template = osTemplate;

    // Set Container name
    veconfig.name = name;

    // Set Container hostname
    veconfig.hostname = hostname;

    // Set Container IP address and netmask.
    ip_addressType[] ip_address = new ip_addressType[1];
    ip_address[0] = new ip_addressType();
    ip_address[0].ip = ip;
    ip_address[0].netmask = netmask;

    // Set network.
    net_vethType[] net = new net_vethType[1];
    net[0] = new net_vethType();
    net[0].host_routed = new object();
    net[0].id = network;
    net[0].ip_address = ip_address;
    veconfig.net_device = net;

    // Set the offline management feature.
    veconfig.offline_managementSpecified = true;
    veconfig.offline_management = offline_management;

    // Finalize the new Container configuration.
    create_input.config = veconfig;

    // Create the Container.
    return env.create(create_input).env.eid;
}
catch (Exception e) {
    return "Exception: " + e.Message;
}
}

```

The function invocation example:

```

createCT("sample_ve", "redhat-as3-minimal", "linux", "x86",
"sample_ve_hostname", "10.16.3.179", "255.255.255.0", "venet0", true
);

```

Getting Server ID From Name

The following is a simple function that will get the Server ID of a Container using its name. This function can be helpful when you want to use any of the other functions that accept the Server ID as a parameter. The reason is that you usually know the name of the Container that you would like to work with, but you most likely don't know its Server ID (the globally unique ID that Agent automatically assigns to every Container).

```

/// <summary>
/// Sample function NameToEid.
/// Gets the Server ID of the Container specified by its name.
/// </summary>
/// <param name="name">Container name.</param>
/// <returns>Server ID of the Container.</returns>
public string NameToEid(string name)
{
    try {
        // Instantiate the proxy class.
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));

        get_info2 getInfo = new get_info2();
        getInfo.eid = new string[1];
        get_list1 velist = new get_list1();
        string eids = "";
        string[] nn = env.get_list(velist);

        foreach (string eid in nn) {
            getInfo.eid[0] = eid;
            envType[] envs = env.get_info(getInfo);
            if (envs.Length != 0) {
                if (env.get_info(getInfo)[0].virtual_config.name ==
name) {
                    eids = eid;
                    break;
                }
            }
            else {
                eids = "";
            }
        }

        return eids;
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```


Starting, Stopping, Restarting a Container

To start a Container, use the `vzaenvmBinding.start()` method passing the Server ID. See [Creating a Simple Client Program](#) (on page 13) for the example on how to obtain the list of the Server IDs.

```

/// <summary>
/// Sample function StartCT.
/// Starts the specified Container.
/// </summary>
/// <param name="ve_eid">The Server ID of the Container.</param>
/// <returns>"OK" or error information.</returns>
public string StartCT(string ve_eid)
{
    try {

        // Instantiate the proxy class.
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));

        // The main input object.
        start start_input = new start();

        // Set Server ID.
        start_input.eid = ve_eid;

        // Start the Container.
        env.start(start_input);

        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```

Stopping and Restarting a Container is similar to the example above. The following two functions demonstrate how it's done.

```

/// <summary>
/// Sample function StopCT.
/// Stops a Container.
/// </summary>
/// <param name="ve_eid">Server ID of the Container.</param>
/// <returns></returns>
public string StopCT(string ve_eid)
{
    try {
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));
        stop1 stop_input = new stop1();

        // Set the Server ID of the Container.
        stop_input.eid = ve_eid;
    }
}

```

```
        // Stop the Container.
        env.stop(stop_input);

        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

/// <summary>
/// Sample function RestartCT.
/// Restarts a Container.
/// </summary>
/// <param name="ve_id">Container Server ID.</param>
/// <returns></returns>
public string RestartCT(string ve_id)
{
    try {
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));
        restart1 restart_input = new restart1();

        // Set the Server ID of the Container.
        restart_input.eid = ve_id;

        // Restart the Container.
        env.restart(restart_input);

        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}
```

Destroying a Container

To destroy a Container, use the `vzaenvmBinding.destroy()` method. The method accepts the Server ID of the Container as a single parameter.

```
/// <summary>
/// Sample function DestroyCT.
/// Destroys a Container.
/// </summary>
/// <param name="ve_eid">Server ID of the Container.</param>
/// <returns>"OK" or error information.</returns>
public string DestroyCT(string ve_eid)
{
    try {
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));
        destroy destroy_input = new destroy();

        // Set the Server ID.
        destroy_input.eid = ve_eid;
        env.destroy(destroy_input);

        return "The Container has been destroyed.";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}
```

Suspending and Resuming a Container

The following two examples show how to suspend and then resume a Virtuozzo Container.

```

/// <summary>
/// Sample function. Suspends a Container.
/// </summary>
/// <param name="ve_eid">The Server ID of the Container.</param>
/// <returns>"OK" or error information.</returns>
public string SuspendCT(string ve_eid)
{
    try {
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));
        suspend1 suspend_input = new suspend1();

        // Set the Container Server ID.
        suspend_input.eid = ve_eid;

        // Suspend the Container.
        env.suspend(suspend_input);

        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

/// <summary>
/// Sample function ResumeCT.
/// Resumes a Container that was previously suspended.
/// </summary>
/// <param name="ve_eid"></param>
/// <returns></returns>
public string ResumeCT(string ve_eid)
{
    try {
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));
        resumel resume_input = new resumel();

        //Set the Container Server ID.
        resume_input.eid = ve_eid;

        // Resume Container.
        env.resume(resume_input);

        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```

Getting Container Configuration Information

A Container configuration information is stored on the Hardware Node. This configuration (also called *virtual configuration*) is used by Virtuozzo Containers to set the necessary Container parameters when the Container is started. To retrieve a Container configuration, use the `vzaenvmBinding.get_info` method. For the complete list and description of the input parameters, see the `vzaenvm/get_info` call in the *Parallels Agent XML Programmer's Reference* guide.

The following sample shows how to retrieve the complete configuration information for the specified Container.

```

/// <summary>
/// Sample function GetConfig.
/// Retrives Container configuration information.
/// </summary>
/// <param name="ve_eid">The Container Server ID.</param>
/// <returns>
/// A string containing the Container configuration information.
/// </returns>
public string GetConfig(string ve_eid)
{
    string ve_info = "";
    try {
        // Instantiate the proxy class.
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));

        // The input parameters.
        get_info2 getInfo_input = new get_info2();
        string[] eids = new string[1];

        // Set the Server ID of the Container for which to get the
info.
        eids[0] = ve_eid;
        getInfo_input.eid = eids;

        // Get the Container information from the Hardware Node.
        envType[] envtype = env.get_info(getInfo_input);

        // Get the Container configuration from the returned object.
        venv_configType veconfig = envtype[0].virtual_config;

        // Get Container name.
        ve_info += "Name: " + envtype[0].virtual_config.name + "\n";

        // Get Container description.
        if (envtype[0].virtual_config.description != null &&
envtype[0].virtual_config.description.Length != 0)
            ve_info += "Description: " +

System.Text.Encoding.ASCII.GetString(envtype[0].virtual_config.descrip
tion) + "\n" +

        //Get network configuration.

```

```

    "Network configuration: \n";
    if (envtype[0].virtual_config.address != null) {
        ve_info += "IP: " + veconfig.address[0].ip + "\n" +
            "Netmask: " + veconfig.address[0].netmask + "\n";
    }

    // Get Container hostname.
    ve_info += "HostName: " + veconfig.hostname + "\n" +
        // Get architecture
        "Architecture: " + veconfig.architecture + "\n" +
        // Get OS
        "OS name: " + veconfig.os.name + "\n" +
        "OS platform: " + veconfig.os.platform + "\n" +
        "OS kernel: " + veconfig.os.kernel + "\n" +
        "OS version: " + veconfig.os.version + "\n" +
        // Get status
        "Status: " + envtype[0].status.state.ToString() + "\n" +
        // Get QoS information.
        "QoS cur: " + veconfig.qos[0].cur.ToString() + "\n" +
        "QoS hard: " + veconfig.qos[0].hard.ToString() + "\n" +
        "QoS id: " + veconfig.qos[0].id + "\n" +
        "QoS soft: " + veconfig.qos[0].soft.ToString(); // + "\n";
    }
    catch (Exception e) {
        ve_info += "Exception: " + e.Message;
    }
    return ve_info;
}

```

Configuring a Container

This section shows how to modify a Container configuration. It is organized into subsections each demonstrating how to modify a particular configuration parameter. The basic idea behind modifying the Container configuration is simple. Agent SOAP API has classes that hold the Container configuration parameters. You instantiate the necessary classes (depending on the parameter type) and populate only those members (configuration parameters) that you would like to modify. You then submit the populated objects to Agent using the appropriate class and method. Upon receiving the new configuration, Agent will updated only those parameters that you specified in the input structure.

Modifying IP Address

Sample Function Parameters:

Name	Description
ve_eid	The Server ID of the Container for which you would like to modify the configuration info.

new_ip	The new IP address. A Virtuozzo Container may have multiple IP addresses assigned to it. When modifying the IP address information, all of the existing address information will be removed from the configuration and the new addresses will be put in their place. In this example, we will be operating with a single IP address for simplicity.
netmask	New netmask.
network	The name of the network interface for which you would like to modify the IP address settings.

Sample Function:

```

/// <summary>
/// Sample function ModifyIP.
/// Modifies the Container IP address.
/// </summary>
/// <param name="ve_eid">The Container Server ID.</param>
/// <param name="new_ip">New IP address.</param>
/// <param name="netmask">New netmask.</param>
/// <param name="network">Network interface name.</param>
/// <returns>"OK" or error information.</returns>
public string ModifyIP(string ve_eid, string new_ip, string netmask,
string network)
{
    try {
        // Instantiate the proxy class.
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));

        // The main input object.
        set2 set_input = new set2();

        // Set the Container Server ID.
        set_input.eid = ve_eid;

        // The Container configuration structure.
        venv_configType1 veconfig = new venv_configType1();

        // Set ip addresses.
        ip_addressType[] ip_address = new ip_addressType[1];
        ip_address[0] = new ip_addressType();
        ip_address[0].ip = new_ip;
        ip_address[0].netmask = netmask;

        // The network interface information structure.
        net_vethType[] net = new net_vethType[1];
        net[0] = new net_vethType();

        // Set the network parameters.
        net[0].host_routed = new object();
        net[0].id = network;
        net[0].ip_address = ip_address;
        veconfig.net_device = net;
        set_input.config = veconfig;

        // Modify the Container configuration.
        env.set(set_input);

        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```


Modifying Hostname

```
/// <summary>
/// Sample function ModifyHostname.
/// Modifies Container hostname.
/// </summary>
/// <param name="ve_eid">The Container Server ID.</param>
/// <param name="new_hostname">New hostname.</param>
/// <returns>OK/Error.</returns>
public string ModifyHostname(string ve_eid, string new_hostname)
{
    try {
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));
        set2 set_input = new set2();

        // Set the container Server ID.
        set_input.eid = ve_eid;

        venv_configType1 veconf = new venv_configType1();

        // Set the new hostname.
        veconf.hostname = new_hostname;
        set_input.config = veconf;

        // Modify the Container configuration.
        env.set(set_input);

        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}
```

Modifying Container Name

```

/// <summary>
/// Sample function ModifyName.
/// Modifies Container name.
/// </summary>
/// <param name="ve_eid">The Container Server ID.</param>
/// <param name="new_name">New Container name.</param>
/// <returns>OK/Error.</returns>
///
public string ModifyName(string ve_eid, string new_name)
{
    try {
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));
        set2 set_input = new set2();

        // Set the Container Server ID.
        set_input.eid = ve_eid;
        venv_configType1 veconf = new venv_configType1();

        // Set new Container name.
        veconf.name = new_name;
        set_input.config = veconf;

        // Modify the Container configuration.
        env.set(set_input);

        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```

Modifying QoS Settings

```

/// <summary>
/// Sample function ModifyQoS.
/// Modifies Container QoS settings.
/// </summary>
/// <param name="ve_eid">The Container Server ID.</param>
/// <param name="qos_id">QoS ID.</param>
/// <param name="hard">New hard limit value.</param>
/// <param name="soft">New soft limit value.</param>
/// <returns></returns>
public string ModifyQoS(string ve_eid, string qos_id, int hard, int
soft)
{
    try {
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));
        set2 set_input = new set2();

        // Set the Container Server ID.
        set_input.eid = ve_eid;

        venv_configType1 veconfig = new venv_configType1();

        // Set Container QoS.
        veconfig.qos = new qosType[1];
        veconfig.qos[0] = new qosType();

        // Set QoS ID.
        veconfig.qos[0].id = qos_id;

        // Set hard limit
        veconfig.qos[0].hardSpecified = true;
        veconfig.qos[0].hard = hard;

        // Set soft limit
        veconfig.qos[0].softSpecified = true;
        veconfig.qos[0].soft = soft;

        // Modify the Container configuration.
        set_input.config = veconfig;
        env.set(set_input);

        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```

Modifying DNS Server Assignment

```

/// <summary>
/// Sample function ModifyDNS.
/// Modifies Container DNS server assignment.
/// </summary>
/// <param name="ve_eid">The Container Server ID.</param>
/// <param name="new_nameserver">New nameserver name.</param>
/// <returns>OK/Error.</returns>
public string ModifyDNS(string ve_eid, string new_nameserver)
{
    try {
        vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(typeof(vzaenvmBinding));
        set2 set_input = new set2();

        // Set the Container Server ID.
        set_input.eid = ve_eid;

        // Container configuration.
        venv_configType1 veconfig = new venv_configType1();

        // Network device.
        veconfig.net_device = new net_vethType[1];
        veconfig.net_device[0] = new net_vethType();

        // Set Container DNS.
        veconfig.net_device[0].nameserver = new string[1];
        veconfig.net_device[0].nameserver[0] = new_nameserver;

        // Modify Container configuration.
        set_input.config = veconfig;
        env.set(set_input);


        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}


```

Cloning a Virtuozzo Container

Cloning refers to a process of creating an exact copy (or multiple copies) of a Virtuozzo Container on the same Hardware Node. The new Container will have its own private area and root directories but the rest of the configuration parameters will be exactly the same. This means that even the parameters that should be unique for each individual Container (IP addresses, hostname, name) will be copied unchanged. You don't have an option to specify new configuration parameter values during the cloning operation. Instead, you will have to clone the Container first and then update the configuration of the new Container in a separate procedure. There are a few exceptions to this rule. You can optionally specify custom private area and root directories for the new Container, but only if you are creating a single copy of the source Container. You also have an option to specify custom Container ID for each clone. If you don't want to set these options manually, their values will be selected automatically.

You can clone both running and stopped Containers. There are some differences when cloning Containers on Windows and on Linux platforms:

 On Linux, the running source Container will be suspended momentarily during the cloning operation. This is done in order to eliminate possible changes to the Container state and status. Once all of the data is read from the source Container, the Container is resumed and the cloning operation proceeds normally.

 On Windows, a snapshot of the Container is taken on the fly, so the source Container operation is never interrupted during cloning.

The following sample illustrates how to clone a Container. The name of the C# class that provides the cloning functionality is `relocatorBinding` (stepping ahead, this class also provides the Container migration functionality, which we'll discuss in the following section). The XML API equivalent of the class is the `relocator` interface.

Sample Function Parameters:

Name	Description
<code>eid</code>	The Server ID of the Container to clone.
<code>count</code>	The number of clones to create.

Sample Function:

```
/// <summary>
/// Sample function CloneCT.
/// Create an exact copy of the specified Container.
/// </summary>
/// <param name="eid">The Server ID of the Source Container.</param>
/// <param name="count">Number of copies to create.</param>
/// <returns>The IDs of the new Virtuozzo Containers.</returns>
///
public string[] CloneCT(string eid, int count)
{
    cloneResponse response;

    try {
        // Instantiate the proxy class
        relocaterBinding relocater =
(relocaterBinding)binder.InitBinding(typeof(relocaterBinding));

        // The main input parameter.
        clone_input = new clone();

        // Set the Server ID of the source Container.
        clone_input.eid = eid;

        // Number of copies to create.
        clone_input.count = 1;

        // Clone the Container(s).
        response = relocater.clone(clone_input);
    }
    catch (Exception e) {
        response = new cloneResponse();
        response.eid_list[0] = "Exception: " + e.Message;
        return response.eid_list;
    }
    return response.eid_list;
}
```

Migrating a Container to a Different Host

You can migrate an existing Container from one Hardware Node to another. The resulting Container is created as an exact copy of the source Container. To migrate a Container, the target Hardware Node must have Virtuozzo Containers software and Parallels Agent installed on it.

The following V2V (virtual-to-virtual) migration types are supported:

- *Offline migration.* Performed on a stopped or a running source Container. If the Container is stopped, all its files are simply copied from the source host to the target host. If the Container is running, the files are first copied to the target machine and then the Container is stopped momentarily. At this point, the data that was copied to the target machine is compared to the original data and the files that have changed since the copying began are updated. The source Container is then started back up. The downtime depends on the size of the Container but should normally take only a minute or so. Offline migration is the default migration type.
- *Simple online migration.* Performed on a running source Container. In the beginning of the migration process, the Container becomes momentarily locked and all of its data, including the states of all running processes, is dumped into an image file. After that, the Container operation is resumed, and the dump file is transferred to the target computer where a new Container is automatically created from it.
- *Lazy online migration.* Instead of migrating all of the data in one big step (as in simple online migration above), lazy migration copies the data over a time period. Initially, only the data that is absolutely necessary to bring the new Container up is copied to the target host. The rest of the data remains locked on the source host and is copied to the destination host on as-needed basis. By using this approach, you can decrease the services downtime to near zero.
- *Iterative online.* During the iterative online migration, the Container memory is transferred to the destination node before the Container data is dumped into an image file. Using this type of online migration allows to attain the smallest service delay.
- *Iterative + lazy online migration.* This type of online migration combines the techniques used in both the lazy and iterative migration types, i.e. some part of the Container memory is transferred to the destination host before dumping a Container, and the rest of the data is transferred on-demand after the Container has been successfully created on the target host.

On successful migration, the original Container will no longer exist on the source node. This is done in order to avoid possible conflicts that may occur if both Containers -- the original and the copy -- are running at the same time. Although the original Container will no longer show up in the Container list on the source node, the Container data will not be deleted. By default, the data is kept in its original location (the Container private area) but the private area directory itself will be renamed. If you wish, you can completely remove the original Container data from the source node by including the `options/remove` parameter in the request.

The name of the C# class that provides the migration functionality is `relocatorBinding`. The XML API equivalent is the `relocator` interface.

The following sample shows how to perform a V2V migration.

Sample Function Parameters:

Name	Description
eid	The source The Container Server ID.
mn_type	Migration type: 0 -- Offline 1 -- Simple online 2 -- Lazy online 3 -- Iterative online 4 -- Iterative lazy online
ip_address	This and the rest of the parameters are the connection and login information that will be used to log in to the target Hardware Node. The target Hardware Node IP address.
port	Port number.
protocol	Communication protocol to use: SSL -- SSL over TCP/IP. TCP -- plain TCP/IP. NamedPipe -- named pipe.
username	User name. The user must have sufficient rights to connect to the target Hardware Node.
realm	Realm ID. The ID of the authentication database against which to authenticate the specified user. In this example, we will be using System Realm -- the user registry of the host operating system.
password	User password.

Sample Function:

```

/// <summary>
/// Sample function Migrate.
/// Migrates a Container to a different Hardware Node.
/// </summary>
/// <param name="eid">Source Container Server ID.</param>
/// <param name="mn_type">Migration type.</param>
/// <param name="ip_address">Target HN IP address.</param>
/// <param name="port">Target HN port number.</param>
/// <param name="protocol">Communication protocol.</param>
/// <param name="username">
/// User name with which to login to the
/// target HN.
/// </param>
/// <param name="realm">
/// Realm ID on the target HN against which to authenticate the user.
/// </param>
/// <param name="password">User password.</param>
/// <returns>"OK" or error information.</returns>
///
public string Migrate(string eid, int migration_type, string
ip_address, uint port, string protocol, string username, string realm,
string password)
{
    try {
        relocaterBinding relocater =
(relocaterBinding)binder.InitBinding(typeof(relocaterBinding));
        migrate_v2v v2v_input = new migrate_v2v();

        // Set the source Container Server ID.
        v2v_input.eid_list = new string[1];
        v2v_input.eid_list[0] = eid;

        /* Set migration type.
        * The "options" member allows you to set other
        * migration options. See Parallels Agent XML Reference
        * for more info.
        */
        v2v_input.options = new v2v_migrate_optionsType();
        v2v_input.options.type = migration_type;

        // Set the target Nardware Node connection info.
        v2v_input.dst = new connection_infoType();
        connection_infoType connection_parm =
(connection_infoType)v2v_input.dst;

        // Set the target Node IP address.
        v2v_input.dst.address = ip_address;

        // Set the port number.
        v2v_input.dst.portSpecified = true;
        v2v_input.dst.port = port;

        // Set protocol.
        v2v_input.dst.protocol = protocol;

        // Set login parameters.
        v2v_input.dst.login = new auth_nameType();

```

```

        v2v_input.dst.login.name =
System.Text.ASCIIEncoding.ASCII.GetBytes(username);
        v2v_input.dst.login.realm = realm;

        // Set user password.
        v2v_input.dst.password =
System.Text.ASCIIEncoding.ASCII.GetBytes(password);

        // Set infinite timeout for the request.
        relocater.Timeout = -1;
        relocater.migrate_v2v(v2v_input);

        return "OK";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```

Performance Monitor

```

}

```

Performance Monitor is an operator that allows to monitor the performance of the Hardware Node and Virtuozzo Containers. By monitoring the utilization of the system resources, you can acquire an important information about your Virtuozzo system health. Performance Monitor can track a range of processes in real time and provide you with the results that can be used to identify current and potential problems. It can assist you with the tracking of the processes that need to be optimized, monitoring the results of the configuration changes, identifying the resource usage bottlenecks, and planning of upgrades.

Agent SOAP API provides the `perf_monBinding` class that allows to retrieve performance reports from the Hardware Node. The types of reports include the performance of the Hardware Node itself and the performances of the individual Virtuozzo Containers. You can select the type and a particular aspect of the server performance that you would like to see. This performance type is called a *class*. The performance aspect is called a *counter*. The following section describes classes and counters in detail.

Classes, Instances, Counters

Performance Class

Performance class is a type of the system resource that can be monitored. This includes CPU, memory, disk, network, etc. A class is identified by its ID. You obtain the IDs of the available classes by retrieving them from the Agent vocabulary. Each performance class is represented by a category in the vocabulary. To distinguish the class categories from other categories, they all belong to another category named `counters`. Different server types (generic, Virtuozzo Containers) have their own sets of performance classes. The following are examples of performance class entries in a vocabulary.

Generic performance class (compatible with physical servers only):

```
<category>
  <id>counters_net</id>
  <category>generic</category>
  <category>counters</category>
  <short>Network usage</short>
  <long>Network usage related parameters</long>
</category>
```

Virtuozzo performance class (compatible with Virtuozzo Containers only):

```
<category>
  <id>counters_vz_net</id>
  <category>virtuozzo</category>
  <category>counters</category>
  <short>Network usage</short>
  <long>Container network-related counters</long>
</category>
```

The following table describes the properties of a performance class:

Property	Description
<code>id</code>	The unique class ID
<code>category</code>	<p>The name of the parent vocabulary category.</p> <p>The <code>counters</code> category indicates that this vocabulary entry is a performance class.</p> <p>The <code>generic</code> category indicates that this class is compatible with generic servers (physical machines or virtual servers treated as real physical servers). Classes compatible with Virtuozzo Containers belong to the <code>virtuozzo</code> category.</p>
<code>short</code>	Short description of the class.
<code>long</code>	Long description of the class.

Class Instance

While class identifies the type of the system resource, the term "instance" refers to a particular device when multiple devices of the same type exist in the system. For example, a network interface in general is a class, but each network card installed in the system is an instance of that class. Each class has at least one instance, but not all classes may have multiple instances.

Performance Counter

Counters are used to measure various aspects of a performance, such as the CPU times, network rates, disk usage, etc. Each class has its own set of counters. Counter data is comprised of the current, minimum, maximum, and average values. You retrieve the list of counters available for a particular class from the vocabulary by specifying the class name as the criteria. Performance counters are stored as *parameters* in a vocabulary. Each counter has a *category* property that contains the name of the performance class this counter belongs to. The following is an example of a performance counter entry in a vocabulary:

```
<parameter>
  <id>counter_cpu_system</id>
  <category>counters_vz_cpu</category>
  <type>int</type>
  <value_type>1</value_type>
  <counter_type>2</counter_type>
  <short>System</short>
  <long>System CPU time</long>
  <measure>seconds</measure>
</parameter>
```

The following table describes the properties of a performance counter:

Property	Description
id	A string containing the unique counter identifier.
category	A string containing the name of the parent performance class (in general, the name of the parent vocabulary category).
type	A string specifying the data type of the counter values. The possible values are: int -- integer. float -- floating point.
value_type	A numeric representation of the counter data type (used internally by Agent).

counter_type	<p>An integer representing the counter type. Depending on the type, the values of the counter can be interpreted differently:</p> <p>0 -- Periodic counter. Contains the minimum, maximum, and average values for the given time period.</p> <p>1 -- Incremental counter. The value of an incremental counter is always higher or equals to the previous value. A good example is a network counter that counts the number of bytes the interface has sent or received. The minimum, maximum, and average values are the same and represent the difference between the current value and the value from the previous report.</p> <p>2 -- Cumulative counter. The minimum, maximum, and average values are the same and represent the total accumulated value since the server was started. On server restart, counter values are reset to zero.</p>
short	Short counter description.
long	Long counter description.
measure	Units of measure (seconds, percent).

Getting a Performance Report

The following lists contain some of the commonly used performance classes and the counters from the `counters_vz_cpu` class as an example. You can get the complete list of classes and their counters from Agent vocabulary. See [Parallels Agent XML Programmer's Reference](#) guide for more information on how to retrieve the performance classes and counters information.

Virtuozzo-specific Performance Classes:

```
counters_vz_cpu
counters_vz_net
counters_vz_loadavg
counters_vz_process
counters_vz_slm
counters_vz_system
counters_vz_memory
counters_vz_hw_net
counters_vz_quota
counters_vz_ubic
```

Counters from the `counters_vz_cpu` class:

```
counter_cpu_system
counter_cpu_user
counter_cpu_idle
counter_cpu_nice
counter_cpu_starvation
counter_cpu_system_states
counter_cpu_user_states
counter_cpu_idle_states
```

The following is an example of two functions working together that retrieve the latest performance report using the specified Server ID, performance class, and performance counter.

The `GetPerfData` sample function initializes and populates the necessary input parameters, gets the performance data from Agent, and then calls the `getData` sample function that extracts the data and puts it into a string that can be displayed on the screen.

Sample Function Parameters:

Name	Description
<code>eid</code>	Server ID of the Container for which to retrieve the performance data.
<code>class_name</code>	The name of the performance class.
<code>counter_name</code>	The name of the performance counter.

Sample Function:

```

/// <summary>
/// Sample function GetPerfData.
/// Gets the Container or the Hardware Node performance data.
/// </summary>
/// <param name="eid"></param>
/// <param name="class_name"></param>
/// <param name="counter_name"></param>
/// <returns>A string containing the performance data.</returns>
///
public string GetPerfData(string eid, string class_name, string
counter_name, string class_instance)
{
    string perf_data = "";
    try {

        // Create binding object.
        perf_monBinding perf_mon =
(perf_monBinding)binder.InitBinding(typeof(perf_monBinding));

        // The main input object.
        get5 get_input = new get5();

        // Set Server ID.
        get_input.eid_list = new string[1];
        get_input.eid_list[0] = eid;

        /* Set the performance class name.
        * Multiple classes can be set if desired.
        */
        get_input.@class = new classType1[1];
        get_input.@class[0] = new classType1();
        get_input.@class[0].name = class_name;

        // Set class instance.
        get_input.@class[0].instance = new classTypeInstance[1];
        get_input.@class[0].instance[0] = new classTypeInstance();
        if (class_instance.Length != 0) {
            get_input.@class[0].instance[0].name = class_instance;
        }

        // Set counter. Multiple counters can be set if desired.
        get_input.@class[0].instance[0].counter = new string[1];
        get_input.@class[0].instance[0].counter[0] = counter_name;

        /* Get the performance data. The returned data is
        * extracted using the GetData helper function, which
        * is defined below.
        */
        GetData(perf_mon.get(get_input), out perf_data);
    }
    catch (Exception e) {
        perf_data += "Exception: " + e.Message;
    }
    return perf_data;
}

/// <summary>
/// Sample function GetData.

```



```

/// This is a helper function that extracts the performance
/// data retrieved by the getPerfData function defined above.
/// </summary>
/// <param name="counters_dat">
/// Contains the data for each class, instance, and counter that
/// were specified in the request that returned this object (the
/// perf_mon.get() call above). To extract the data, we have to
/// iterate through all
/// of them.
/// </param>
/// <param name="counters_info">
/// Output. A string containing the extracted data.
/// </param>
///
public void GetData(perf_dataType[] counters_dat, out string
counters_info)
{
    counters_info = "";
    if (counters_dat.Length != 0) {
        foreach (perf_dataType counter_dat in counters_dat) {
            if (counter_dat.@class != null) {
                foreach (perf_dataTypeClass dat in counter_dat.@class)
                {
                    counters_info += "\n Class name: " + dat.name +
"\n" +
                    "Instances:\n";
                    if (dat.instance != null) {
                        foreach (perf_dataTypeClassInstance instance
in dat.instance) {
                            counters_info += " DataClassInstance: " +
instance.name + "\n";
                            if (instance.counter != null) {
                                foreach
(perf_dataTypeClassInstanceCounter counter in instance.counter) {
                                    counters_info += " \nName:" +
counter.name + "\n" +
                                    " avg: " +
counter.value.avg + "\n" +
                                    " cur: " +
counter.value.cur + "\n" +
                                    " max: " +
counter.value.max + "\n" +
                                    " min: " +
counter.value.min;
                                }
                            }
                        }
                    }
                    else {
                        counters_info += " No counters." +
"\n";
                    }
                }
            }
            else {
                counters_info += "No instances." + "\n";
            }
        }
    }
    else {
        counters_info += "No classes." + "\n";
    }
}

```

```

        counters_info += "Intervals:\n" +
            "Start time: " + counter_dat.interval.start_time + "\n" +
            "End time: " + counter_dat.interval.end_time + "\n" +
            "EID: " + counter_dat.eid + "\n";
    }
}
else {
    counters_info += "No data returned.";
}

```

Monitoring Alerts

```

}

```

Alerts are notifications that report the system resource allocation problems such as approaching or exceeding certain limits. Alerts are usually used for monitoring of the Container health, predicting its performance, or collecting information that can be used to optimize the Container performance. Use the `alertmBinding` class to check if a Container has alerts of any kind currently raised and to retrieve the alert data if it does.

The alert levels are described in the table below.

Alert level	ID	Description
Green	0	Normal operation. This alert is raised when one of the higher-level alerts is canceled.
Yellow	1	Moderately dangerous situation. The specified parameter is coming close (within 10%) to its soft limit barrier.
Red	2	Critical situation. The parameter exceeded its soft limit or came very close to the hard limit. Depending on the parameter type, either some process can be killed at any time now, or the next resource allocation request can be refused.
Black	3	The worst-case scenario. The hard limit was reached, the requested resource allocation was refused or some process overusing the resource was killed. Once raised, the black alert remains in effect for 5 minutes.

A Virtuozzo Container may have multiple alerts raised at any given time. The following function demonstrates how you can check if a Container has any alerts currently raised, and to retrieve the alert information if it does. The function accepts the list of Containers for which to check and retrieve the alert information.

```

/// <summary>
/// Sample function GetAlerts.
/// Retrieves the system alert information for the specified
/// Container.
/// </summary>
/// <param name="ve_eid">
/// Server ID of the Container to get the alerts for.
/// </param>
/// <returns>A string containing the alert information.</returns>
///
public string GetAlerts(string[] ve_eid)
{
    string list_result = "";
    try {
        // Instantiate the proxy class
        alertmBinding alertm =
(alertmBinding)binder.InitBinding(typeof(alertmBinding));

        // The main input object.
        get_alerts get_alerts_input = new get_alerts();

        // Set Container list.
        get_alerts_input.eid_list = ve_eid;

        // Get the alert information.
        foreach (eventType al_event in
alertm.get_alerts(get_alerts_input)) {
            list_result += "Data: \n";

            // Get the alert data.
            resource_alertType res_data =
(resource_alertType)al_event.data.event_data;
            // Read the alert data.
            list_result += "  Class: " + res_data.@class + "\n" +
                // Get counter.
                "    Counter: " + res_data.counter + "\n" +
                // Get eid.
                "    Eid: " + res_data.eid + "\n" +
                // Get instance.
                "    Instance: " + res_data.instance + "\n" +
                // Get type.
                "    Type: " + res_data.type.ToString() + "\n" +
                // Get current value.
                "    Cur: " + res_data.cur + "\n" +
                // Get hard limit.
                "    Hard: " + res_data.hard + "\n" +
                // Get soft limit.
                "    Soft: " + res_data.soft + "\n" +
                // Get event name.
                "Name: " + al_event.info.name + "\n" +
                // Get count.
                "Count: " + al_event.count.ToString() + "\n" +
                // Get event category.
                "Category: " + al_event.category + "\n" +
                // Get event message.

```

```

        "Message: " +
        System.Text.ASCIIEncoding.ASCII.GetString(al_event.info.message) +
        "\n" +
        // Get parameters
        "Parameters: ";
        /* Call the helper function to extract the
        * event message parameter values.
        */
        GetParams(al_event.info.parameter, ref list_result);
    }
}
catch (Exception e) {
    list_result += "Exception: " + e.Message;
}
return list_result;
}

/// <summary>
/// Sample function GetParams.
/// This is a helper function that extracts the
/// alert message parameter values.
/// </summary>
/// <param name="parameter">The name of the parameter.</param>
/// <param name="list">
/// Output. Values.
/// </param>
///
void GetParams(infoType[] parameter, ref string list)
{
    string ss = " ";

    foreach (infoType param in parameter) {
        list += ss + "Message: " +
        System.Text.ASCIIEncoding.ASCII.GetString(param.message) +
        "    Info name: " + param.name + "\n";

        if (param.parameter != null) {
            GetParams(param.parameter, ref list);
        }
    }
}

```

File Management

Agent SOAP API provides the `filerBinding` class that can be used in client applications to manage files and directories on Hardware Nodes and Virtuozzo Containers, including listing files and directories, uploading, downloading, copying, moving and removing, searching, etc. In this section, we will demonstrate how to perform some of the most common file management operations.

Request Routing

Before you delve into the details of the individual file management operations, we have to discuss an important Agent API feature called *Request Routing*.

Most of the Virtuozzo-specific methods have the Server ID (*eid*) input parameter which is used to specify the Container on which the operation should be performed. For example, when you start or stop a Container, you pass Server ID as an input parameter (see **Starting, Stopping, Restarting a Container** (on page 33)). In contrast, methods of the classes that allow to perform operations on both Virtuozzo Containers and Hardware Nodes don't usually have this parameter. For example, the `filerBinding.list` method (lists files and directories) does not have the *eid* parameter. So how do you get file listing for a particular Container? That's where request routing comes in.

Routing

You can tell Agent to route the request to the specified Container and execute it there instead of executing it on the Hardware Node level. You accomplish this by including the `dst/host` (destination host) parameter in the Agent request message header to contain the Server ID of the target Virtuozzo Container. By not including the `dst/host` parameter in the message header, you are instructing Agent to perform the operation on the Hardware Node.

We already have a sample class called `Binder` (see page 15) that implements this functionality. The `InitBinding` method of the `Binder` class instantiates a proxy class and populates the request message header. The overloaded `InitBinding` method accepts the Container Server ID as a second parameter and adds the `dst/host` parameter to the header. To route a request to a particular Virtuozzo Container, use this method to instantiate a proxy class passing the target Server ID to it.

The following sample code shows how to create a proxy class object without the request routing information specified.

```
vzaenvmBinding env =
(vzaenvmBinding)binder.InitBinding(sizeof(vzaenvmBinding));
```

The following sample uses the request routing feature. The requests initiated by the class methods will be routed to the specified Virtuozzo Container.

```
string ve_eid = "3b8f950a-981d-b94d-bde1-647df39674f1";
filerBinding filer =
(filerBinding)binder.InitBinding(sizeof(filerBinding), ve_eid);
```

So the question is, when exactly do you use the request routing feature? The rule of thumb is as follows:

- If a method that you want to use to perform an operation on a Virtuozzo Container doesn't accept Server ID (*eid*) as a parameter, you have to use request routing.
- If the method has the *eid* parameter, don't use request routing. If you try to route a request to a Container by mistake, it most likely will fail with a message saying that the functionality is not supported.

There are only a few classes that rely on the request routing functionality when the target of an operation is a Virtuozzo Container. Here's the list:

Class name	Description
<code>computermBinding</code>	Computer management. Provides methods for managing Hardware Nodes and Containers as if they were regular physical machines.
<code>filerBinding</code>	Provides methods for managing files and directories.
<code>firewallmBinding</code>	Firewall management (Linux only).
<code>processmBinding</code>	System processes management. Provides methods for executing a program inside a Container and for killing system processes.
<code>servicemBinding</code>	Services management. Provides methods for managing OS system services.
<code>usermBinding</code>	Provides methods for managing users and groups on Hardware Nodes and Virtuozzo Containers.

Listing Files

The following sample shows how to get a list of files and directories from the Hardware Node. The path parameter is used to specify the directory (or multiple directories) for which to get the list of files and subdirectories.

```

/// <summary>
/// Sample function ListHNfiles.
/// Lists files and directories on a Hardware Node.
/// </summary>
/// <param name="path">Pathname(s).</param>
/// <returns>A string containing the list of files.</returns>
///
public string ListHNfiles(string[] path)
{
    try {
        string list_result = "";

        // Instantiate the proxy class.
        filerBinding filer =
(filerBinding)binder.InitBinding(typeof(filerBinding));

        // The main input object.
        list2 list = new list2();

        // Set pathnames.
        byte[][] paths = new byte[path.Length][];
        for (int i = 0; i < path.Length; i++) {
            paths[i] =
System.Text.ASCIIEncoding.ASCII.GetBytes(path[i]);
        }
        list.path = paths;

        /* Get file listing, then iterate through it and
        * populate a string variable with the results.
        */
        foreach (FileType file in filer.list(list)) {
            list_result += "\n\nName: " +
System.Text.ASCIIEncoding.ASCII.GetString(file.name) +
            "\nSize: " + file.size.ToString() + "\nType: " +
file.type.ToString();
        }
        return list_result;
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```

The following sample functions shows how to get a list of files and directories from a Virtuozzo Container. Note that the only difference between this function and the sample function above is how we create the proxy class object. Here, we use the overloaded binder.InitBinding method that has the additional eid parameter, which we use to specify the target Virtuozzo Container. As a result, the request will be routed to the specified Container and the file listing will be obtained from the Container instead of the Hardware Node.

```

/// <summary>

```

```

/// Sample function ListCTfiles.
/// Lists files and directories on a Container.
/// </summary>
/// <param name="path">Pathname(s).</param>
/// <param name="eid">
/// Server ID of the Container.
/// </param>
/// <returns>A string containing the list of files.</returns>
///
public string ListCTfiles(string[] path, string eid)
{
    try {
        string list_result = "";

        // Instantiate the proxy class.
        filerBinding filer =
(filerBinding)binder.InitBinding(typeof(filerBinding), eid);

        // The main input object.
        list2 list = new list2();

        // Set pathnames.
        byte[][] paths = new byte[path.Length][];
        for (int i = 0; i < path.Length; i++) {
            paths[i] =
System.Text.ASCIIEncoding.ASCII.GetBytes(path[i]);
        }
        list.path = paths;

        /* Get file listing, then iterate through it and
        * populate a string variable with the results.
        */
        foreach (fileType file in filer.list(list)) {
            list_result += "\n\nName: " +
System.Text.ASCIIEncoding.ASCII.GetString(file.name) +
"\nSize: " + file.size.ToString() + "\nType: " +
file.type.ToString();
        }
        return list_result;
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```


Uploading a File

The `filerBinding` class provides the `upload` method for uploading files to the Hardware Node or a Virtuozzo Container. The maximum block size that you can upload in a single method invocation is 512 kilobytes. If uploading a file larger than 512K, you'll have to read the file data in 512K or smaller blocks and transfer each one individually, i.e. one method invocation per data block. The first method invocation creates a file on the destination server and adds the first data block to it. The rest of the source data can be appended to or inserted at the desired position in the destination file. The following sample shows how to upload a file larger than 512K in size.

```

/// <summary>
/// Sample function UploadFile.
/// Shows how to upload a file to a Virtuozzo Container.
/// </summary>
/// <param name="source">Source file name and path.</param>
/// <param name="destination">
/// Destination file name and path.
/// </param>
/// <param name="eid">Server ID of the target Container.</param>
/// <returns>"OK" or error info.</returns>
/// <example>
/// <para>
/// Example: UploadFile(@"c:\share\packets.txt",@"/tmp/pack15")
/// </para>
/// </example>
///
public string UploadFile(string source, string destination, string
eid)
{
    try {
        filerBinding filer =
(filerBinding)binder.InitBinding(typeof(filerBinding), eid);

        /* An offset in the destination file at which
        * to insert the data. The value of -1 means
        * to append the data to the file.
        */
        int offset = -1;

        // Set the read buffer size (in bytes).
        // The maximum possible block size is 512K.
        long readBlockSize = 524288;

        // The main input object.
        upload upload_input = new upload();

        /* You can upload more than one file in a single call.
        * Here we upload just one file.
        */
        upload_input.file = new uploadFile[1];

        // Initialize the wrapper for the source file path.
        FileInfo info = new FileInfo(source);

        // Open the file for reading.

```

```

        FileStream fstream = new FileStream(source, FileMode.Open,
FileStream.Read);

        // Read the file data in 512K blocks.
        long remaining_bytes;
        while ((fstream.Position < fstream.Length)) {

            remaining_bytes = fstream.Length - fstream.Position;

            // Initialize the buffer for the current data block.
            byte[] readBlock;
            if (remaining_bytes > readBlockSize) {
                readBlock = new byte[readBlockSize];
            }
            else {
                readBlock = new byte[remaining_bytes];
            }

            // Read the data block into the buffer.
            fstream.Read(readBlock, 0, readBlock.Length);

            /* When adding data to an existing file,
             * the "force" option must be set. Otherwise, the
             * upload will fail.
             */
            upload_input.force = new object();

            // Set the offset at which to start writing.
            upload_input.file[0] = new uploadFile();
            upload_input.file[0].offsetSpecified = true;
            upload_input.file[0].offset = offset;

            // Set the destination file name and path.
            upload_input.file[0].path =
System.Text.ASCIIEncoding.ASCII.GetBytes(destination);

            // The block of data to write in this iteration.
            upload_input.file[0].body = readBlock;
            upload_input.file[0].size = readBlock.Length;

            // Upload the data block.
            filer.upload(upload_input);
        }

        fstream.Close();
        return "OK!";
    }
    catch (Exception e) {
        return "Exception: " + e.Message;
    }
}

```

Downloading a File

Downloading a file from the Hardware Node or a Virtuozzo Container is similar to uploading a file described in the previous section. To download a file, you must specify the file name and path on the source server. For files larger than 512K, the file data must be transferred in 512K (or less) data blocks. The received data can be written to a file on the client machine or processed any other way that your client application may require.

Index

A

Agent SOAP API • 8

C

Certificates Policy Preparation • 14
 Choosing Development Project • 10
 Classes, Instances, Counters • 51
 Cloning a Virtuozzo Container • 45
 Complete Program Code • 22
 Configuring a Container • 38
 Connection URL • 18
 Creating a Simple Client Program • 13
 Creating a Virtuozzo Container • 28

D

Destroying a Container • 35
 Development Platforms • 8
 Documentation Conventions • 5
 Downloading a File • 67

E

Errors and Resolution • 12

F

Feedback • 6
 File Management • 60

G

General Conventions • 6
 Generating Proxy Classes From WSDL • 11
 Getting a Performance Report • 54
 Getting Container Configuration Information • 37
 Getting Server ID From Name • 32

I

Installation • 9
 Instantiating Proxy Classes • 15
 Introduction • 7

L

Listing Files • 63
 Logging in and Creating a Session • 18

M

Main Program File • 14
 Migrating a Container to a Different Host • 47
 Modifying Container Name • 42
 Modifying DNS Server Assignment • 44
 Modifying Hostname • 41
 Modifying IP Address • 38
 Modifying QoS Settings • 43
 Monitoring Alerts • 58

P

Performance Monitor • 50
 Preface • 5

R

Request Routing • 61
 Retrieving a List of Virtuozzo Containers • 20

S

Shell Prompts in Command Examples • 6
 Starting, Stopping, Restarting a Container • 33
 Suspending and Resuming a Container • 36

T

Typographical Conventions • 5

U

Uploading a File • 65

V

Virtuozzo Container Management • 27

W

What is Parallels Agent? • 7