
SWsoft

Virtuozzo SDK C++ Tutorial

1.0



SWSOFT™

(c) 1999-2007

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

Copyright © 1999-2007 by 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	4
Documentation Conventions.....	4
Typographical Conventions.....	4
Shell Prompts in Command Examples	5
General Conventions	5
Feedback	5
Glossary	6
Creating Your First Client Program	8
1. Creating the Project	8
2. Connecting to VZAgent.....	9
3. Getting a List of Environments.....	11
4. Creating a New Environment.....	21
5. Starting, Stopping, Restarting an Environment.....	26
6. Getting Environment Configuration	28
7. Modifying Environment Configuration	33
8. Destroying an Environment	36
9. The Complete Program Code.....	37
Adding Environment Monitoring	55
1. Retrieving a List of Performance Counters.....	56
2. Getting a Single Performance Report	61
3. Receiving Periodic Performance Reports	63
4. Receiving System Event Notifications	67
5. Getting a List of Current Alerts	71
6. Receiving Alerts by Subscription	74
Request Redirection	77
1. Managing Files and Directories	79
2. Managing Operating System Services	85
Appendix A Sample Program	89
Index	119

CHAPTER 1

Preface

In This Chapter

Documentation Conventions.....	4
Feedback	5

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 QoS 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 virtuo- 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.

Glossary

VZAgent is a server-side software that enables systems running it to be managed remotely from client applications. VZAgent gives the ability to manage physical computers and virtual machines of various types.

Access Point is an object which serves as a point of entry to the existing Environment structure through VZAgent server. Upon establishing a connection with the server, Access Point collects information about the Environments and then stores it in a cache repository that it creates on the client machine. The information in the cache repository is kept current through automatic updates, and can be quickly retrieved at any time at user request.

Virtualization Technology is a server virtualization software product that allows to run multiple virtual machines on a single physical host. Some of the examples are Virtuozzo and VMware.

Environment is a physical or a virtual machine running in the *VZAgent cluster*. For example, a physical machine hosting VZAgent software is an Environment. A Virtuozzo Virtual Environment hosted by that server is also an Environment.

Environment ID (EID) is a universally unique identifier assigned to every Environment in VZAgent cluster at the time the Environment is added to the cluster.

Virtual Environment ID (VEID) -- Virtualization products may assign their own IDs to virtual Environments. For example, Virtuozzo Virtual Environments have IDs, which are selected by the user at the time a VE is created (the ID can be any number greater than 100). Most of the VZAgent API calls use *EID* when performing a task on an Environment. Some virtualization technology-specific calls may use *VEID* to access a virtual machine from the "inside" of the virtualization product. Unlike EIDs, VEIDs are not guaranteed to be globally unique. In case of Virtuozzo, a VEID is unique only in the context of a single Virtuozzo installation.

VZAgent Cluster is a term specific to VZAgent and refers to a network of Environments each running its own VZAgent software and interconnected with each other by means of internal VZAgent mechanisms. The Environments in a cluster are organized in a hierarchical structure where there's one *Master Environment* and many *Slave Environments*. VZAgent installed in a Master Environment administers the entire cluster by allocating, monitoring, and controlling cluster resources. Master VZAgent is capable of accessing any slave Environment in a cluster, which means that the client program connected to the master VZAgent can send requests to any of the slave Environments in the cluster.

Parent Environment is a physical or virtual machine that hosts other virtual Environments. The machine, therefore, is said to be a Parent Environment to the virtual Environments that it hosts. Note that virtual Environments may host other virtual Environments -- theoretically speaking, the number of levels of the hierarchy is not limited. In such a hierarchy, each virtual Environment has just one Parent Environment.

Root Environment is the Environment that the client program is connected to. It could be any type of Environment -- virtual or physical -- and it can be located anywhere in the cluster hierarchy. A client can normally access the root Environment and all its child Environments. If the root Environment is also the master Environment in a cluster, the program can access the entire cluster.

CHAPTER 2

Creating Your First Client Program

This part of the tutorial walks you step-by-step through creating a Windows console application that uses the most common VZAgent functionality. The application that you are going to create, will be able to connect to VZAgent server, log in a user, and then perform some of the most common VZAgent tasks such as getting a list of Environments, creating an Environment, starting and stopping an Environment, getting an Environment information, and others.

In This Chapter

1. Creating the Project.....	8
2. Connecting to VZAgent.....	9
3. Getting a List of Environments	11
4. Creating a New Environment.....	21
5. Starting, Stopping, Restarting an Environment.....	26
6. Getting Environment Configuration	28
7. Modifying Environment Configuration	33
8. Destroying an Environment	36
9. The Complete Program Code.....	37

1. Creating the Project

To create the initial project, follow these steps:

- 1 In the Visual Studio development environment, click **New** on the **File** menu, and then click **Project**.
- 2 Click the **Visual C++ Projects** folder and select **Win32 Console Project**.
- 3 Type `VzSDK-C++Tutorial` as the project name.
- 4 Enter the location that you would like to use for your project files (or use the default location) and click **OK**.
- 5 The **Win32 Application Wizard** opens. Simply click **Finish** without changing any of the default settings. The wizard will create the new project and will add some files to it by default.
- 6 In **Solution Explorer** right-click on `VzSDK-C++Tutorial` project and select **Properties** from the pop-up menu.
- 7 The **VzSDK-C++Tutorial Property Pages** windows opens. Modify the following properties:

Project property	New value
C/C++/Code Generation > Runtime Library	Multi-threaded Debug DLL (/MDd)
C/C++/Language > Enable Run-Time Type Info	Yes (/GR)

C/C++/Precompiled Headers > Create/Use Precompiled Header	Not Using Precompiled Headers
Linker/Input > Additional Dependencies	VZLFunctionalityAgent.lib VZLCore.lib VZLLibCore.lib VZLFunctionalityCore.lib VZLConnection.lib VZLLibAgent.lib VZLFunctionality.lib VZAFunctionalityAgent.lib VZAFunctionality.lib

Click OK when done.

Your project is now set up and you are ready to start writing the application.

2. Connecting to VZAgent

The first task to be done in any VZAgent client application is establishing a connection with VZAgent server.

In the Visual Studio development environment, open VzSDK-C++Tutorial.cpp file. Add the following #include directive to it.

```
#include <VZLLibAgent/VZLAccessPoint.h>
```

Add the following using namespace directives.

```
// Standard namespace.
using namespace std;

// VZL namespace. Contains common classes.
using namespace VZL;

// VZA namespace. Contains Virtuozzo-specific classes.
using namespace VZA;
```

Add a global Access Point object variable. We use global variable here because the rest of the functions in our sample program will be using it to perform their tasks.

```
VZLAccessPointSP accessPoint;
```

You are now ready to add the code that will establish a connection with VZAgent server. The connection procedure consists of the following steps:

- 1 Populating the VZLConnectionInfo structure with the connection parameters (substitute the values provided here with the ones appropriate to your VZAgent server installation).

```
// VZAgent server IP address.
string address = "192.168.0.44";

// Login.
// "vzadmin" is the default VZAgent administrator account.
string login = "vzadmin";

// Password.
string password = "mypass";

// Create the Connection Info object.
VZLConnectionInfo connInfo(address, login, password);
```

2 Create the Access Point object.

```
accessPoint = VZLAccessPointPrototype::createInstance();
```

3 Establish a connection with VZAgent server.

```
if (accessPoint->initializeSync(connInfo,
    NULL, CACHE_ALL | CACHE_TREE) != 0)
{
    cout << "Error, unable to connect to VZAgent. ";
    cout << "Please check the connection parameters." << endl;

    // Get the error information.
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error ( " << errorData.code << " ) ";
    cout << errorData.message << endl;
    return -1;
}
```

On initial connection, the Access Point object will cache some (or all) of the Environment information. The options that you pass to the `accessPoint->initializeSync()` method determine how the information will be cached. In this example we use the following options:

CACHE_ALL : cache all Environment data, including the list of Environments, Environment configuration information, vocabulary. This option also starts Environment monitoring.

CACHE_TREE : cache the data for the entire Environment tree (as opposed to the root Environment only) and monitor all Environments in the tree.

You can build and run your program now. If the `accessPoint->initializeSync()` call completes without error, continue to the next step of this tutorial. If you are receiving an error do the following:

- 1 Make sure that VZAgent is running by going to your server and typing the following at the command prompt and then pressing ENTER:

```
vzagent_ctl status
```

The output produced by this command will be displayed on the screen informing you of the current VZAgent status. If VZAgent is not running, use the following command to start it:

```
vzagent_ctl start
```

If you are still having problems, try reinstalling VZAgent software.

- 2 Make sure that you can access the server that hosts VZAgent via the LAN, then double check the connection parameters that you've entered in the previous step of the tutorial, and try again.

3. Getting a List of Environments

In the previous step, you connected to VZAgent server and populated the Access Point cache with Environment data. The next logical step might be to retrieve the list of Environments, and to display it to the user, so the user can browse it and select the Environment that he/she wants to work with.

In the VZAgent infrastructure, any Environment (virtual or physical) can host other virtual Environments. Therefore, the Environment list is actually a tree (possibly, a multi-level tree). The tree will contain at least the root Environment (the Environment that hosts VZAgent server that you are connected to). If the root Environment hosts virtual Environments (such as Virtuozzo VEs), the tree will contain them too. If you have VZAgent cluster set up and are connected to VZAgent running in Master Environment, the tree will contain all Environments from the entire cluster.

To use the examples contained in the rest of this section, your `#include` list should contain the following:

```
#include "stdafx.h"
#include <tchar.h>
#include <iostream>
#include <VZLLibAgent/VZLAccessPoint.h>
#include <VZAFunctionality/VZAEEnvM.h>
```

The Environment information that you retrieved from the server is contained in the `VZLEnvCache::EnvValuesList` object. The object is initialized and populated with data at the time you establish a connection with VZAgent (see [Step 2: Connecting to VZAgent](#)). A smart pointer to the object is defined in the `VZLAccessPointPrototype` class and can be obtained using the following code:

```
VZLAccessPointPrototype::EnvValuesListSP envList;
envList = accessPoint->getEnvList();
```

where `accessPoint` is the Access Point object that we declared globally and initialized earlier.

The `VZLEnvCache::EnvValuesList` is an STL-like container. To examine its contents, you will need an iterator declared as follows:

```
VZLEnvCache::EnvValuesList::const_iterator cit;
```

The elements of the container are `VZLEnv` objects, each containing an individual Environment information. You obtain a pointer to each `VZLEnv` object by positioning the iterator at the required element of the container. You can then use the iterator's member access operator `->()` to access the `VZLEnv` class members. For example, the following code will iterate through the entire `envList` container, and will display a flat list of Environments.

```
// Regular configuration structure.
VZLEnvConfig envConfig;

// Virtuozzo virtual config structure.
VZAEEnvConfig vzEnvConfig;

// List of the Environment IP addresses.
```

```

VZLEnvConfig::IPAddressList IPList;

// Environment type.
VZLEnvConfigBasic::type_t envType;

// Environment name.
string envName;
// Hostname
string envHostname;
// IP address.
string IPaddress;

for (cit = envList->begin(); cit != envList->end(); ++cit)
{
    // Based on the Environment type, use the
    // correct configuration structure and then
    // read the Environment configuration info
    // from it.
    envType = cit->getType();

    // Generic environment.
    if (envType == "generic")
    {
        envConfig = cit->getConfig();
        envName = envConfig.getName().c_str();
        envHostname = envConfig.getHostname().c_str();
        envConfig.getAddresses(IPList);
        if (IPList.isSpecified() && IPList->size() != 0)
        {
            IPaddress = IPList->begin()->ip;
        }
    }
    // Virtuozzo Virtual Environment.
    else if (envType == "virtuozzo")
    {
        vzEnvConfig = cit->getVirtualConfig();
        envName = vzEnvConfig.getName().c_str();
        envHostname = vzEnvConfig.getHostname().c_str();
        vzEnvConfig.getAddresses(IPList);
        if (IPList.isSpecified() && IPList->size() != 0)
        {
            IPaddress = IPList->begin()->ip;
        }
    }
    else
    {
        // This is a placeholder for future types.
        // If you have other Environment types,
        // use the correct config structure here.
    }

    // Display the basic Environment properties
    // on the screen:

    // EID (globally unique Environment ID)
    cout << cit->getEID().toString() << " ";

    // Environment name.
    cout << envName << " ";
}

```

```

// Hostname.
cout << envHostname << " ";

// IP address (an Environment may actually have
// multiple IP addresses).
cout << IPaddress << " ";

// Environment type.
cout << envType.c_str() << endl;
}

```

The code above displays a flat list of Environments, but since you know that the Environments are organized into a tree (and if your goal is to display the Environments to the user as a tree) then you might want to enhance this code. For example, you can call the function recursively to build each individual branch of the tree beginning with the root Environment. The complete example program at the end of this section demonstrates how it can be accomplished.

One other important point that the code above demonstrates is that depending on the Environment type (i.e. generic, virtuoizzo, etc.) different objects are used to store the Environment configuration information, so you must use the correct object type and a correct method that returns the information of a particular type. In the code example above, we retrieve the generic Environment information from cache using the `VZLEnvConfig` object and its `getConfig` method:

```

VZLEnvConfig envConfig;
envConfig = cit->getConfig();

```

On the other hand, the Virtuozzo VE information is retrieved using the `VZAEnvConfig` object:

```

VZAEnvConfig vzEnvConfig;
vzEnvConfig = cit->getVirtualConfig();

```

Other Environment types may use their own classes to store the configuration information. Check the appropriate plugin documentation for the available configuration classes. A plugin-specific class will be inherited from the `VZLVEnvConfig` class, which in turn is inherited from the `VZLEnvConfig` class.

As you should already know, an Environment is identified by a globally unique ID (EID), which is assigned to it at the time the Environment is created. All operations on a particular Environment are performed by referencing it through EID. As you can see from the main code example above, the EID can be obtained using the `getEID` method while iterating through the Environment information container (the `envList` object in our example).

Example Program:

This subsection contains a complete console program that you can copy and paste into the `VzSDK-C++Tutorial.cpp` file that you created earlier. The program performs the following tasks:

- Connects to VZAgent server.
- Builds and displays the Environment tree using information contained in the Access Point cache.
- Allows to select an Environment from the list and then use the selected Environment's EID in other calls.

The program uses only the features that have already been discussed in the previous steps of this tutorial.

```

// VzSDK-C++Tutorial.cpp : Defines the entry point for the
// console application.
//

#include "stdafx.h"
#include <tchar.h>
#include <iostream>
#include <VZLLibAgent/VZLAccessPoint.h>
#include <VZAFunctionality/VZAEvm.h>

using namespace std;
using namespace VZL;
using namespace VZA;

// Access Point object.
VZLAccessPointSP accessPoint;

// EID list.
typedef pair<eid_t, string> EIDNamePair;
typedef vector<EIDNamePair> EIDNameList;

string getEnvStatusDesc(const VZLEnvStatus& envStatus);
int displayEnvTree(VZLAccessPointPrototype::EnvValuesListSP envList,
    eid_t parentEID, EIDNameList & eids, int indent);
EIDNamePair selectEnvFromTree();

int _tmain(int argc, _TCHAR* argv[])
{
    // VZAgent server IP address.
    string address = "10.16.0.15";

    // Login.
    string login = "vzadmin";

    // Password.
    string password = "mypass";

    // Create the Connection Info object.
    VZLConnectionInfo connInfo(address, login, password);

    // Create the Access Point object.
    accessPoint = VZLAccessPointPrototype::createInstance();

    // Establish a connection with VZAgent server.
    if (accessPoint->initializeSync(connInfo,
        NULL, CACHE_ALL | CACHE_TREE) != 0)
    {
        cout << "Error, unable to connect to VZAgent." << endl;
        cout << "Please check the connection parameters." << endl;
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error (" << errorData.code << ") ";
        cout << errorData.message << endl;
        return -1;
    }

    EIDNamePair eidNamePair;
    eidNamePair.first = accessPoint->getRootEID();
    eidNamePair.second = "root";

```

```

    eidNamePair = selectEnvFromTree();

    cout << endl << endl;
    cout << "You selected: " << endl << endl;
    cout << "Environment ID (EID): " << eidNamePair.first.toString()
<< endl;
    cout << "Environment Title:      " << eidNamePair.second << endl;

    return 0;
}

// Here we have two functions that build the Environment tree.
// One function will actually build and display the
// tree (it will be called recursively).
// The second function will initially call the first
// function and will allow the user to select
// an Environment from the tree.

// Builds and displays the Environment tree.
// Parameters
// envList :   The Environment list retrieved from
//              the Access Point cache.
// parentEID : Parent Environment ID (the top level
//              of this branch).
// eids :      The list of the retrieved EID/Name pairs.
// indent :    An indentation at which the Environment name
//              should be displayed.

int displayEnvTree(VZLAccessPointPrototype::EnvValuesListSP
                  envList, eid_t parentEID,
                  EIDNameList& eids, int indent)
{
    // The Environment container iterator.
    // The iterator's member access operator ->() returns
    // a pointer to the VZLEnv object, containing
    // an individual Environment information.
    VZLEnvCache::EnvValuesList::const_iterator cit;

    // Regular configuration structure.
    VZLEnvConfig envConfig;

    // Virtuozzo virtual config structure.
    VZAEnvConfig vzEnvConfig;

    // List of the Environment IP addresses.
    VZLEnvConfig::IPAddressList IPList;

    // Environment type.
    VZLEnvConfigBasic::type_t envType;

    // Environment name.
    string envName;
    // Hostname
    string envHostname;
    // IP address.
    string IPaddress;
    // Environment title.
    string envTitle;

    // Iterate through the Environment container.

```

```

for (cit = envList->begin(); cit != envList->end(); ++cit)
{
    // The function actually builds one tree branch
    // at a time. Each Environment in the list has
    // the parent EID property, which indicates the
    // parent Environment EID. You can use this property
    // to determine whether an Environment belongs to
    // the current branch. If does, add it to the branch.
    if (cit->getParentEID() == parentEID)
    {
        // some tree formatting...
        cout << string(indent, ' ') << "|" << endl;
        cout << string(indent, ' ') << "|-- ";

        // Based on the Environment type, use the
        // correct configuration structure and then
        // read the Environment configuration info
        // from it.
        envType = cit->getType();

        // Generic environment.
        if (envType == "generic")
        {
            envConfig = cit->getConfig();
            envName = envConfig.getName().c_str();
            envHostname = envConfig.getHostname().c_str();
            envConfig.getAddresses(IPList);
            if (IPList.isSpecified() && IPList->size() != 0)
            {
                IPaddress = IPList->begin()->ip;
            }
        }
        // Virtuozzo Virtual Environment.
        else if (envType == "virtuozzo")
        {
            vzEnvConfig = cit->getVirtualConfig();
            envName = vzEnvConfig.getName().c_str();
            envHostname = vzEnvConfig.getHostname().c_str();
            vzEnvConfig.getAddresses(IPList);
            if (IPList.isSpecified() && IPList->size() != 0)
            {
                IPaddress = IPList->begin()->ip;
            }
        }
        else
        {
            // This is a placeholder for future types.
            // If you have other Environment types,
            // use the correct config structure here.
        }

        // There are several fields that can be used
        // as the Environment title. None of those
        // fields are mandatory, however. Let's check
        // each field and use as the Environment title
        // the one that actually has a value.
        envTitle = envName;
        if (envTitle.empty())
        {
            // Hostname

```



```

        envTitle = envHostname;
        if (envTitle.empty())
        {
            // IP address
            envTitle = IPaddress;
            if (envTitle.empty())
                envTitle = "N/A";
        }
    }

    // Insert the current EID/Name pair into
    // the list. The list will be used later
    // to determine the Environment title based on
    // the ID (for display purposes).
    EIDNamePair pair;
    pair.first = cit->getEID();
    pair.second = envTitle;
    eids.push_back(pair);

    // Display the basic Environment properties
    // on the screen.
    // Title:
    cout << "(" << eids.size() - 1 << ")" << " " << envTitle;

    // some screen output formatting...
    char buffer[30];
    itoa((eids.size() - 1), buffer, 10);
    cout << string((33 - (strlen(buffer) +
envTitle.length()))), ' ');

    // Environment type:
    cout << envType.c_str() << "\t";

    // Environment status.
    cout << getEnvStatusDesc(cit->getStatus()) << endl;

    // Go to the next level of recursion to
    // build the next tree level.
    indent = indent + 3;
    displayEnvTree(envList, cit->getEID(), eids, indent);
    indent = indent - 3;
    }
}
return 0;
}

// The second function (works together with
// displayEnvTree() listed above)
// Allows to select an Enviroment from the tree.
EIDNamePair selectEnvFromTree()
{
    // Get the Environment list from Access Point cache.
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();

    // Display heading.
    cout << "Environments\t\t\t\t\tType\t\tStatus" << endl;
    cout << string(65, '-') << endl;

    // Get the root Environment information.

```

```

VZLEnvCSP env = envList->getEnv(accessPoint->getRootEID());

// A list of EIDs.
EIDNameList eids;
// The EID/Name pair.
EIDNamePair eidNamePair;
// Popoulate the pair with root Environment values.
eidNamePair.first = accessPoint->getRootEID();
eidNamePair.second = "root";
eids.push_back(eidNamePair);
cout << "(0) root - " <<
    env->getConfig().getHostname().c_str() << endl;

// Build and display the Environment tree,
// starting with the root Environment.
// The function will be called recursively for each
// Environment in the list, because each Environment
// may have child Environments of its own.
displayEnvTree(envList, accessPoint->getRootEID(), eids, 0);

// Allow the user to select an environment from
// the list. The user must select the Environment
// index number (a sequential number added to
// each Environment entry by the displayEnvTree() function).
// The Environment EID/Name pair is the element of the
// eids array with the selected index.
int iIn;
while (true)
{
    cout << endl;
    cout << "Enter Environment index number: ";
    cin >> iIn;

    if (iIn >= eids.size() || iIn < 0)
    {
        cout << "Invalid entry, please try again." << endl;
        iIn = 0;
        continue;
    }

    return eids[iIn];
}

// A helper function to convert the Environment state and
// transition codes into a description that can be displayed
// to the user. The Environment state and transition are
// enumerations defined in the VZAgent client library.
string getEnvStatusDesc(const VZLEnvStatus& envStatus)
{
    string statusDesc;

    // Determining the VE state and transition.
    if (envStatus.transition == ENV_TRANS_NONE)
    {
        // VE is not in transition. Display status.
        switch (envStatus.state)
        {
            case ENV_STAT_UNEXIST:
                statusDesc = "dropped";

```

```

        break;
    case ENV_STAT_DOWN:
        statusDesc = "stopped";
        break;
    case ENV_STAT_MOUNTED:
        statusDesc = "mounted";
        break;
    case ENV_STAT_SUSPENDED:
        statusDesc = "suspended";
        break;
    case ENV_STAT_RUNNING:
        statusDesc = "running";
        break;
    default:
        statusDesc = envStatus.state;
        break;
    }
}
else
{
    // VE is in transition (its stable state is changing).
    switch (envStatus.transition)
    {
        case ENV_TRANS_CREATING:
            statusDesc = "creating ...";
            break;
        case ENV_TRANS_MOUNTING:
            statusDesc = "mounting ...";
            break;
        case ENV_TRANS_STARTING:
            statusDesc = "starting ...";
            break;
        case ENV_TRANS_STOPPING:
            statusDesc = "stopping ...";
            break;
        case ENV_TRANS_DESTROYING:
            statusDesc = "destroying ...";
            break;
        case ENV_TRANS_BACKING_UP:
            statusDesc = "backing up ...";
            break;
        case ENV_TRANS_RESTOREING:
            statusDesc = "restoring ...";
            break;
        case ENV_TRANS_SUSPENDING:
            statusDesc = "suspending ...";
            break;
        case ENV_TRANS_RESUMING:
            statusDesc = "resuming ...";
            break;
        default:
            statusDesc = envStatus.transition;
    }
}

return statusDesc;
}

```

Output

If you build and run the program now, you should see the output similar to the following example:

Environments	Type	Status
(0) root - hw15		
-- (1) VE1157120736781	virtuozzo	running
-- (2) VE1157126285203	virtuozzo	stopped
-- (4) VE1157124478812	virtuozzo	stopped
-- (5) VE1157117936453	virtuozzo	suspended
-- (6) VE1157130232281	virtuozzo	running
-- (7) VE1157120389250	virtuozzo	stopped
-- (8) VE1157132118093	virtuozzo	stopped

Enter Environment index number: 4

You selected:

Environment ID (EID): 17a9316a-c890-4070-a853-94eb505a0916
 Environment Title: VE1157124478812

The first entry in the list is the root Environment (hostname hw15). The other entries are Virtuozzo VEs hosted by it. To select an Environment from the list, enter its index number (the numeric value in the parentheses).

4. Creating a New Environment

In this step, we will add a function that creates a new Virtuozzo Virtual Environment. The function will work as a wizard -- it will ask the user a series of questions allowing to select the necessary Environment configuration parameters, and then, as the last step, will create an Environment using the selected options.

First, update your `#include` list to contain the following directives:

```
#include "stdafx.h"
#include <tchar.h>
#include <iostream>
#include <VZLLibAgent/VZLAccessPoint.h>
#include <VZLFunctionality/VZLLibFunctionality.h>
#include <VZLFunctionalityAgent/VZLFunctionalityAgentMain.h>
#include <VZLFunctionality/VZLFunctionalityMain.h>
#include <VZAFFunctionality/VZAFFunctionalityMain.h>
#include <VZAFFunctionalityAgent/VZAFFunctionalityAgentMain.h>
#include <VZAFFunctionality/VZAEnvM.h>
#include <VZAFFunctionality/VZAPkgM.h>
#include <VZLFunctionality/VZLEnvSampleM.h>
```

Now declare the new function as follows:

```
int createVirtuozzoVE();
```

- 1 First, you have to create the Environment management object. Every virtualization technology plugin has its own Environment management interface inherited from the base `VZLEnvMBase` interface. In this tutorial, we will work with Virtuozzo VEs, so you will need the `VZAEnvM` object, which is the Virtuozzo-specific implementation. To create the object and obtain a smart pointer to it, use the object factory exactly as shown in the following example:

```
// Initialize the VZA (Virtuozzo) plugin.
initVZAFFunctionalityAgent();

// Create the Environment Management object using object factory.
VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
    VZLLibFunctionality::kit().getEnvM(
        accessPoint->getRootEID(),
        accessPoint,
        VZA::vzaFunctionalityType));

if (!vzaenvm)
{
    cerr << "ERROR, unable to create VZA::VZAEnvMSP object." << endl;
    return -1;
}
```

- 2 The next step is to create the Environment configuration information object (you will populate it with data in the next steps). Again, each virtualization technology plugin uses its own class to store the Environment configuration. In this case, we are using the Virtuozzo-specific class (see the descendants of the `VZLEnvConfig` class for other implementations).

```
VZAEEnvConfig envConf;
```

- 3** You now have to populate the `envConf` object with the new Environment configuration data. There's a number of ways to create a Virtuozzo VE (refer to Virtuozzo documentation for more information). In this example, we will take one of the most common approaches. First, you have to select the VEID (Virtuozzo-level Environment ID), Environment name, hostname, IP address, and the Environment description.

```
int iIn;
string sIn;
```

Virtuozzo Virtual Environment ID (VEID).

```
cout << "VEID: ";
cin >> iIn;
envConf.setVeid(iIn);
```

Environment name.

```
cout << "Environment Name: ";
cin >> sIn;
envConf.setName(sIn);
```

IP address (you can modify the code to allow the user to enter multiple IP addresses).

```
cout << "IP address: ";
cin >> sIn;
VZLEnvConfig::IPAddressList::value_type IPList;
VZLIPAddress IPAddress;
IPAddress.ip = sIn;
IPList.insert(IPAddress);
envConf.setAddresses(IPList);
```

Hostname.

```
cout << "Hostname: ";
cin >> sIn;
envConf.setHostname(sIn);
```

Environment description.

```
cout << "Description: ";
cin >> sIn;
envConf.setDescription(sIn);
```

- 4** Next, you have to select the OS template.

First, get the list of the installed standard OS templates.

```

cout << endl << "Available OS templates:" << endl;

VZA::VZATEMSP vzaTem = VZALibFunctionality::kit().getVZATEM(
    accessPoint->getRootEID(),
    accessPoint,
    VZA::vzaFunctionalType);

if (!vzaTem)
{
    cerr << "ERROR, unable to create VZATEMSP object." << endl;
    return -1;
}

VZAPackageInfoList pkgInfoList;
eid_t eid;

```

Passing empty eid, which means that we want the templates that are installed in the root Environment.

```

if (vzaTem->ls(eid, &pkgInfoList, "os") != 0)
{
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error (" << errorData.code << ") ";
    cout << errorData.message << endl << endl;
}

bool bPkgListEmpty = true;
std::vector<string> pkgList;
int ctr = 0;

if (pkgInfoList.size() != 0)
{
    for (VZAPackageInfoList::iterator pit = pkgInfoList.begin();
        pit != pkgInfoList.end(); ++pit)
    {
        pkgList.push_back(pit->name);
        cout << "(" << ctr++ << ") ";
        cout << pit->name << "\tstandard template" << endl;
    }
}

```

Now, get the list of the installed EZ templates.

```

VZA::VZAPkgMSP vzapkgm = dynamic_pointer_cast<VZAPkgM>(
    VZLLibFunctionality::kit().getPkgM(
        accessPoint->getRootEID(),
        accessPoint,
        VZA::vzaFunctionalType));

if (!vzapkgm)
{
    cerr << "ERROR, unable to create VZAPkgMSP object." << endl;
    return -1;
}

VZLPackageList pkgOut;
vector<string> pkgType;
pkgType.push_back("os");
VZLPackageManagerListOptions pkgOptions;
pkgOptions.types = pkgType;

vzapkgm->list(accessPoint->getRootEID(),
             &pkgOut, NULL, pkgOptions);

if (pkgOut.size() == 0 && ctr == 0)
{
    cout << "No OS templates were found." << endl;
    system("pause");
    return -1;
}

for (VZLPackageList::iterator pit = pkgOut.begin();
     pit != pkgOut.end(); ++pit)
{
    pkgList.push_back((*pit)->name);
    cout << "(" << ctr++ << " ) ";
    cout << (*pit)->name << "\tEZ Template" << endl;
}

```

5 Allow the user of the program to select the desired OS template from the list.

```

cout << endl;
cout << "Enter template number: ";
cin >> iIn;

```

Set the OS template that the user selected.

```

envConf.setOsTemplate(VZATemplate(pkgList[iIn]));

```

6 In this step, we retrieve the list of the available sample configurations, display them on the screen and allow the user to select the desired configuration.

Create the VZLEnvSampleM object (sample configuration management) using the object factory.


```

initVZLFunctionalityAgent();
VZLEnvSampleMSP vzlSampleConf =
    VZLLibFunctionality::kit().getEnvSampleM(
        accessPoint->getRootEID(),
        accessPoint);

if (!vzlSampleConf)
{
    cerr << "ERROR, unable to create VZLEnvSampleMBaseSP object." <<
endl;
    return -1;
}
Retrieve a list of the available sample configurations.
VZLSampleID sampleConfigID;
VZLSampleConfList confs;
VZLSampleIDList ids;
vzlSampleConf->getSampleConf(&confs, ids);

pair<VZLGUID, string> IdNamePair;
vector<pair<VZLGUID, string> > sampleList;
cout << endl << "Sample configuration list" << endl << endl;

```

Iterate through the container and display the list on the screen.

```

ctr = 0;
for (VZLSampleConfList::iterator sit = confs.begin();
    sit != confs.end(); ++sit)
{
    IdNamePair.first = sit->id;
    IdNamePair.second = sit->name;
    sampleList.push_back(IdNamePair);
    cout << "(" << ctr++ << " ) ";
    cout << sit->name << endl;
}

```

Allow the user to select the desired configuration.

```

cout << "Enter sample number: ";
cin >> iIn;
envConf.setBaseSampleId(sampleList[iIn].first);

```

7 The Environment configuration information object is now populated with required data. The final step is to create the VE.

```

eid_t envIDout;

if (vzaenvm->create(&envIDout, envConf) != 0)
{
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error ( " << errorData.code << " ) ";
    cout << errorData.message << endl;
    return -1;
};
cout << "VE created successfully." << endl;

return 0;

```

5. Starting, Stopping, Restarting an Environment.

Starting, stopping, or restarting an Environment is both simple and straightforward. The only thing you have to remember is that you cannot perform these operations on the Environments in certain statuses. For example, you cannot start an Environment that is already running (obviously), or an Environment that is in some of the transition statuses. Similarly, you cannot start, stop, or restart an Environment that is currently suspended, or is in the "repair" mode (see `VZLEnvMBase::suspend` and `VZLEnvMBase::repair` methods for more info).

One other thing to consider is that when you read the Environment status information from the Access Point cache, the status may no longer be valid by the time you initiate start, stop, or restart operation, so the operation may also fail. To avoid a situation like that, refresh the information that you have on display or in your own memory frequently (or when possibly needed) using the information contained in the Access Point cache. The Access Point cache is synchronized with the server-side automatically on a periodic basis, or when an event that requires synchronization occurs on the server side.

- 1 To start, stop, or restart an existing virtual Environment, first create the environment management object. Once again, you must use the class that is appropriate to the type of the Environment that you want to work with.

```
// Initialize the VZA (Virtuozzo) functionality plugin.
initVZAFunctionalityAgent();

// Create the Virtuozzo Environment Management
// object using the object factory.
VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
    VZLLibFunctionality::kit().getEnvM(
        accessPoint->getRootEID(),
        accessPoint,
        VZA::vzaFunctionalType));

if (!vzaenvm)
{
    cerr << "ERROR, unable to create VZA::VZAEnvMSP object." << endl;
    return -1;
}
```

- 2 To start a VE, use the `VZAEnvM::start` method passing the Environment ID to it.

```
if (vzaenvm->start(eid) != 0)
{
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error ( " << errorData.code << " ) ";
    cout << errorData.message << endl;
    return -1;
}

cout << endl << "VE started successfully." << endl;
return 0;
```

- 3 To stop a VE, use the `VZAEnvM::stop` method.

```
if (vzaenvm->stop(eid) != 0)
{
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error ( " << errorData.code << " ) ";
    cout << errorData.message << endl;
    return -1;
}
```

4 To restart a VE, use the `VZAEvm::restart` method.

```
if (vzaenvm->restart(eid) != 0)
{
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error ( " << errorData.code << " ) ";
    cout << errorData.message << endl;
    return -1;
}
```

6. Getting Environment Configuration

The Environment configuration information is contained in the Access Point cache. You already accessed this data partially when you retrieved the Environment list earlier in this tutorial. This section will demonstrate how to retrieve the extended Environment information for different Environment types.

Determining Environment type

First, let's create a function that will determine the type of the selected Environment. We need this information because configuration information is retrieved differently for different Environment types.

```
int displayEnvInfo(eid_t eid)
{
    // Get the Environment list from cache.
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();

    // Get the specified Environment information.
    VZLEnvCSP env = envList->getEnv(eid);

    // Get the VZLEnvConfig object containing the specified
    // Environment configuration info.
    VZLEnvConfig envConfig = env->getConfig();

    // Call the appropriate function to extract and display the
    // configuration info on the screen.
    if (env->getType() == "generic")
    {
        // Displays the generic Environment info.
        displayGenericEnvInfo(eid);
    }
    else if (env->getType() == "virtuozzo")
    {
        // Displays Virtuozzo VE info.
        displayVirtuozzoVEinfo(eid);
    }
    else
    {
        // This is a placeholder for future types.
        cout << "Unsupported Environment type." << endl;
        return -1;
    }

    return 0;
}
```

The function above accepts the EID of the Environment to get the information for. You can use the code example listed in [Step 3: Getting a List of Environments](#) (see page 11) to select the Environment and to get its EID. We haven't implemented the `displayGenericEnvInfo()` and `displayVirtuozzoEnvInfo()` functions yet. Let's do it now.

Getting the generic Environment info

To see what kind of info you can retrieve for a generic Environment, take a look at the `VZLEnvConfig` class. The class has a collection of `get/set` methods that will give you an idea of what the regular Environment configuration is comprised of.

- 1 Create a new function and declare it as follows (the body of the function is described in the rest of this subsection):

```
int displayGenericEnvInfo(eid_t eid);
```

- 2 The first thing that you have to do is obtain a pointer to the `EnvValuesList` object containing the information for all known Environments.

```
VZLAccessPointPrototype::EnvValuesListSP envList;  
envList = accessPoint->getEnvList();
```

- 3 The `envList` container contains `VZLEnv` objects, each holding a complete information about an individual Environment. If you remember, when we retrieved the Environment list earlier in this tutorial, we used the iterator to access the individual `VZLEnv` objects. There's another way to access these objects, and we are going to use it here. We can actually obtain a pointer to the `VZLEnv` object containing the information about a specific Environment if we know its EID:

```
VZLEnvCSP env = envList->getEnv(eid);
```

Now that we have a pointer to the `VZLEnv` object, we can get the `VZLEnvConfig` object and use its `get/set` methods to get (or set) the Environment configuration information.

```
VZLEnvConfig envConfig = env->getConfig();
```

Basic Environment information.

```
cout << endl << "Environment Information" << endl << endl;  
cout << "EID:\t\t" << eid.toString() << endl << endl;  
cout << "Name: \t\t" << envConfig.getName().c_str() << endl;  
cout << "Hostname:\t" << envConfig.getHostname().c_str() << endl;  
cout << "Domain:\t" << envConfig.getDomain().c_str() << endl;  
cout << "Type:\t\t" << env->getType().c_str() << endl;  
cout << "Status:\t\t" << getEnvStatusDesc(env->getStatus()) << endl;  
cout << "Architecture:\t" << envConfig.getArchitecture().c_str() << endl;  
cout << "Description:\t" << envConfig.getDescription().c_str() << endl;
```

Operating system information.

```
cout << endl << "OS:" << endl;  
cout << "    Name: " << envConfig.getOS().name.c_str() << endl;  
cout << "    Platform: " << envConfig.getOS().platform.c_str() << endl;  
cout << "    Version: " << envConfig.getOS().version.c_str() << endl;  
cout << "    Kernel: " << envConfig.getOS().kernel.c_str() << endl;
```

IP address list.

```

VZLEnvConfig::IPAddressList Iplist;
envConfig.getAddresses(Iplist);

cout << endl << "IP Addresses:" << endl;
for (VZLEnvConfig::IPAddressList::value_type::iterator cit =
    Iplist->begin(); cit != Iplist->end(); ++cit)
{
    cout << "    " << cit->ip << ", " << cit->netmask;
}
cout << endl;
return 0;

```

Output

The screen output produced by this function should look similar to the following example:

```

Environment Information

EID:                94d0509d-alec-474a-91c2-4464d9dd360f

Name:               hw15
Hostname:           hw15
Domain:             mydomain
Type:               generic
Status:             running
Architecture:       i686
Description:        Hardware Node 15

OS:
  Name: Red Hat Enterprise Linux AS release 4 (Nahant Update 3)
  Platform: Linux
  Version:
  Kernel: 2.6.9-023stab021.2-smp

IP Addresses:
  10.16.0.15, 10.16.0.20

```

Getting the Virtuozzo Environment info

To see what kind of info you can retrieve for a Virtuozzo Virtual Environment, take a look at the `get/set` methods of the `VZAEnvConfig` class and its parent class `VZLVEEnvConfig` (the base virtual configuration class).

- 1 Create a new function and declare it as follows (the body of the function is described in the rest of this subsection):

```
int displayVirtuozzoVEinfo(eid_t eid);
```

- 2 The first few steps, when retrieving the Virtuozzo VE configuration, are exactly the same as when retrieving the generic Environment configuration. You obtain the `VZLAccessPointPrototype::EnvValuesListSP` pointer and then get a pointer to the `VZLEnv` object containing the information for the specified Environment.

```

VZLAccessPointPrototype::EnvValuesListSP envList;
envList = accessPoint->getEnvList();
VZLEnvCSP env = envList->getEnv(eid);

```

The `eid` parameter used in the call above is the EID of the Environment that we want to get the information for. For example, the Environment that the user selected from the list.

- 3** The next step is to get the `VZAEEnvConfig` object containing the specified Virtuozzo VE configuration information. In order to do that, you have to use the `VZLEnv::getVirtualConfig` method (instead of `VZLEnv::getConfig` as was the case when you were retrieving the generic Environment info).

```
VZAEEnvConfig envConfig = env->getVirtualConfig();
```

- 4** You can now use the `envConfig` object to get the VE configuration info. The information includes many of the same properties as the generic Environment configuration but adds properties that are specific to the virtual environments in general, and Virtuozzo VEs in particular.

Basic Environment information.

```
cout << endl << "Environment Information" << endl << endl;
cout << "EID:\t\t" << eid.toString() << endl << endl;
cout << "Name: \t\t" << envConfig.getName().c_str() << endl;
cout << "Hostname:\t" << envConfig.getHostname().c_str() << endl;
cout << "Domain:\t" << envConfig.getDomain().c_str() << endl;
cout << "Type:\t\t" << env->getType().c_str() << endl;
cout << "Status:\t\t" << getEnvStatusDesc(env->getStatus()) << endl;
cout << "Architecture:\t" << envConfig.getArchitecture().c_str() << endl;
cout << "Description:\t" << envConfig.getDescription().c_str() << endl;
```

Operating system information.

```
cout << endl << "OS:" << endl;
cout << "    Name: " << envConfig.getOS().name.c_str() << endl;
cout << "    Platform: " << envConfig.getOS().platform.c_str() << endl;
cout << "    Version: " << envConfig.getOS().version.c_str() << endl;
cout << "    Kernel: " << envConfig.getOS().kernel.c_str() << endl;
```

IP address list.

```
VZLEnvConfig::IPAddressList Iplist;
envConfig.getAddresses(Iplist);

cout << endl << "IP Addresses:" << endl;
for (VZLEnvConfig::IPAddressList::value_type::iterator cit =
    Iplist->begin(); cit != Iplist->end(); ++cit)
{
    cout << "    " << cit->ip << ", " << cit->netmask;
}
}
```

Virtuozzo-specific settings.

```
cout << endl;
veid_t veid;
envConfig.getVeid(veid);
cout << "VEID: " << veid << endl;
```

The on-boot parameter (determines whether the VE should be started on system boot).

```
VZLBoolOptionalProperty bOn;
envConfig.getOnBoot(bOn);
cout << "On-boot: " << (bOn ? "on" : "off") << endl;
```

The offline management parameter. Determine whether the offline management is turned on for the VE (the offline management allows to manage a Virtuozzo VE even if it's not running).

```
envConfig.getOfflineManagement(bOn);
cout << "Offline mgmt: " << (bOn ? "on" : "off") << endl;
```

The VE root directory.

```
string rootDir;
envConfig.getVERoot(rootDir);
cout << "Root directory: " << rootDir << endl;
```

Network interfaces.

```
VZAEEnvConfig::NetVethListAttribute netVethList;
envConfig.getNetVeths(netVethList);
VZAEEnvConfig::NetVethListAttribute::value_type::iterator iter =
netVethList->begin();

cout << "Network interfaces: " << endl;
for (iter; iter != netVethList->end(); ++iter)
{
    cout << iter->toString() << ", ";
}
```

Output

The screen output produced by this function should look similar to the following example:

```
Environment Information:

EID:                47e37bae-510b-4f71-ac05-36f451d9a575

Name:               VE-200
Hostname:           VE200-host
Domain:             mydomain
Type:               virtuozzo
Status:             running
Architecture:
Description:        Test VE.

OS:
  Name: redhat-as3-minimal
  Platform: Linux
  Version: 20060907
  Kernel: 2.6.9-023stab021.2-smp

IP Addresses:
  10.16.1.15

VEID:               200
On-boot:            on
Offline mgmt:       on
Root directory:     /vz/root/1
Network interfaces: eth0
```


7. Modifying Environment Configuration

As you should already know, VZAgent distinguishes two types of Environment configuration -- regular and virtual. Virtual Environments have both configurations, while generic Environments usually have only the regular configuration.

When you are making modifications to the virtual configuration, the new values for some of the parameters are applied to the Environment right away (this results in the regular Environment configuration change as well), but some parameters may require restarting of the Environment for the changes to take effect (see the documentation for your virtualization product for details).

When you are modifying the regular configuration, the virtual configuration will not be synchronized with it anymore, so the next time you reboot your virtual Environment, the virtual configuration settings will replace the values that you previously modified. Normally, if you are working with a virtual Environment, you should modify the virtual configuration information only. The rest of the material in this section describes how it is accomplished.

Let's create a new function and call it `modifyVirtualConfig`. The function will accept the Environment ID, determine its virtualization technology, and will call the appropriate function to perform the modifications.

```
int modifyVirtualConfig(eid_t eid)
{
    // Get the Environment info from the Access Point cache.
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();
    VZLEnvCSP env = envList->getEnv(eid);

    // Virtuozzo VE.
    if (env->getType() == "virtuozzo")
    {
        modifyVirtuozzoVEConfig(env);
    }
    // Normally, there's no virtual configuration for
    // generic Environments.
    else if (env->getType() == "generic")
    {
        cout << endl;
        cout << "Error: not a virtual Environment, unable to
continue.";
        cout << endl;
        return -1;
    }
    // Some other, currently unsupported virtualization technology.
    else
    {
        // This is a placeholder for future types.
        cout << "Unsupported virtual Environment type." << endl;
    }

    return 0;
}
```

We haven't implemented the `modifyVirtuozzoVEConfig` function yet. Let's do it now. The function will modify the specified Virtuozzo VE virtual configuration. Declare the function as follows.

```
int modifyVirtuozzoVEConfig(VZLEnvCSP env);
```

The function accepts the `VZLEnvCSP` smart pointer to the `VZLEnv` object containing the selected Environment information. The following describes the code that goes into the function's body.

Get the Environment ID and the current virtual configuration information.

```
eid_t eid = env->getEID();
VZAEEnvConfig envConfigOld = env->getVirtualConfig();
```

Declare another virtual configuration object that you will use to store the new configuration parameter values.

```
VZAEEnvConfig envConfigNew;
```

The following code provides a simple user interface allowing the user to enter the new values. The interface displays the current parameter value and allows the user to type in the new value. In this demonstration, we modify only some of the parameters, which are specific to the Virtuozzo VEs. Other configuration parameters can be populated with the new values in a similar manner.

VE name.

```
string sIn;
cout << endl << "Modify configuration" << endl << endl;
cout << "Current name: " << envConfigOld.getName().c_str() << endl;
cout << "New Name: ";
cin >> sIn;
envConfigNew.setName(sIn);
```

Hostname.

```
cout << endl;
cout << "Current hostname: " << envConfigOld.getHostname().c_str() << endl;
cout << "New Hostname: ";
cin >> sIn;
envConfigNew.setHostname(sIn);
cout << endl;
```

The on-boot option. This parameter controls whether the VE will be started automatically on host machine boot.

```
VZLBoolOptionalProperty bOn;
envConfigOld.getOnBoot(bOn);
cout << "On-boot current: " << (bOn ? "on" : "off") << endl;
cout << "On-boot (0 = off, 1 = on): ";
cin >> sIn;
envConfigNew.setOnBoot((sIn == "1" ? true : false));
```

The offline management parameter. Specifies whether the offline management for this VE is turned on. Offline management allows to manage a VE even if it is not currently running.

```
envConfigOld.getOfflineManagement(bOn);
cout << "Offline mgmt current: " << (bOn ? "on" : "off") << endl;

cout << "Offline management (0 = off, 1 = on): ";
cin >> sIn;
```

```
envConfigNew.setOfflineManagement(sIn == "1" ? true : false);
```

Once the user is done entering the new values, you are ready to submit the new configuration information to the VZAgent server. First, we have to create the Environment management object using object factory.

```
VZA::VZAEvmMSP vzaenvm = dynamic_pointer_cast<VZAEvmM>(
    VZLLibFunctionality::kit().getEnvM(
        accessPoint->getRootEID(),
        accessPoint,
        VZA::vzaFunctionalType));

if (!vzaenvm)
{
    cerr << "ERROR, unable to create VZA::VZAEvmMSP object." << endl;
    return -1;
}
```

Submit the envConfigNew object, containing the new Environment information, to VZAgent.

```
if (vzaenvm->set(eid, envConfigNew) != 0)
{
    // Error. Retrieve the error information
    // and display it on the screen.
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error (" << errorData.code << ") ";
    cout << errorData.message << endl;
    return -1;
}
```

The modifyVirtuozzoEnvInfo function returns 0 if the operation was a success.

```
cout << endl;
return 0;
```

8. Destroying an Environment

This function destroys the specified Virtuozzo VE.

```
int destroyVirtuozzoVE(eid_t eid)
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAFunctionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "ERROR, unable to create VZA::VZAEnvMSP object."
              << endl;
        return -1;
    }
    if (vzaenvm->destroy(eid) != 0)
    {
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error ( " << errorData.code << " ) ";
        cout << errorData.message << endl;
        return -1;
    }

    cout << endl << "VE was destroyed." << endl;
    return 0;
}
```

9. The Complete Program Code

This section provides a complete sample console application that includes the functionality that we discussed in the tutorial so far. The application features a simple console user interface that will allow you to run individual code examples. We are not suggesting that you should use it in your real-life applications, but for the purpose of this tutorial, it provides a convenient way for you to examine the different features that the tutorial demonstrates.

The interface provides a menu from which the user can select the task to perform. Here's what the menu looks like when displayed on the screen:

```
=====
MAIN MENU
=====
Selected Environment: root

0.  Display this menu
1.  Display Environment list
2.  Create new Environment
3.  Start
4.  Stop
5.  Restart
6.  Get Environment info
7.  Modify virtual configuration
8.  Destroy Environment

Enter menu item number (any alpha key to exit):
```

The complete program code.

```
#include "stdafx.h"
#include <tchar.h>
#include <iostream>
#include <iomanip>
#include <VZLLibAgent/VZLAccessPoint.h>
#include <VZLFunctionality/VZLLibFunctionality.h>
#include <VZLFunctionalityAgent/VZLFunctionalityAgentMain.h>
#include <VZLFunctionality/VZLFunctionalityMain.h>
#include <VZLFunctionality/VZLEnvSampleM.h>
#include <VZAFFunctionality/VZAFFunctionalityMain.h>
#include <VZAFFunctionalityAgent/VZAFFunctionalityAgentMain.h>
#include <VZAFFunctionality/VZAEvm.h>
#include <VZAFFunctionality/VZAPkgM.h>
#include <VZAFFunctionality/VZALibFunctionality.h>

using namespace std;
using namespace VZL;
using namespace VZA;

// Access Point object.
VZLAccessPointSP accessPoint;

// EID list.
typedef pair<eid_t, string> EIDNamePair;
typedef vector<EIDNamePair> EIDNameList;
```

```

string getEnvStatusDesc(const VZLEnvStatus& envStatus);
int displayEnvTree(VZLAccessPointPrototype::EnvValuesListSP envList,
eid_t parentEID, EIDNameList & eids, int indent);
EIDNamePair selectEnvFromTree();
int createVirtuozzoVE();
int startVirtuozzoVE(eid_t eid);
int stopVirtuozzoVE(eid_t eid);
int restartVirtuozzoVE(eid_t eid);
int destroyVirtuozzoVE(eid_t eid);
int displayEnvInfo(eid_t eid);
int displayGenericEnvInfo(eid_t eid);
int displayVirtuozzoVEinfo(eid_t eid);
int modifyVirtualConfig(eid_t eid);
int modifyVirtuozzoVEConfig(VZLEnvCSP env);
int destroyVirtuozzoVE(eid_t eid);

////////////////////////
// Main Program

int _tmain(int argc, _TCHAR* argv[])
{
    // VZAgent server IP address.
    string address = "192.168.0.151";

    // Login.
    string login = "vzadmin";

    // Password.
    string password = "1q2w3e";

    // Create the Connection Info object.
    VZLConnectionInfo connInfo(address, login, password);

    // Create the Access Point object.
    accessPoint = VZLAccessPointPrototype::createInstance();

    // Establish a connection with VZAgent server.
    if (accessPoint->initializeSync(connInfo,
        NULL, CACHE_ALL | CACHE_TREE) != 0)
    {
        cout << "Error: unable to connect to VZAgent." << endl;
        cout << "Please check the connection parameters." << endl;
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error (" << errorData.code << ") ";
        cout << errorData.message << endl;
        return -1;
    }

    // Initialize the generic functionality plugin.
    initVZLFunctionalityAgent();

    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAFFunctionalityAgent();

    EIDNamePair eidNamePair;
    eidNamePair.first = accessPoint->getRootEID();
    eidNamePair.second = "root";

```

```

/*****
// Main program loop

int in;
while (true)
{
    cout << endl;
    cout << string (65, '=') << endl << endl;
    cout << "MAIN MENU" << endl << endl;
    cout << string (65, '=') << endl;
    cout << "Selected Environment: " << eidNamePair.second <<
endl;
    cout << endl;
    cout << " 0.  Display this menu" << endl;
    cout << " 1.  Display Environment list" << endl;
    cout << " 2.  Create new Environment" << endl;
    cout << " 3.  Start" << endl;
    cout << " 4.  Stop" << endl;
    cout << " 5.  Restart" << endl;
    cout << " 6.  Get Environment info" << endl;
    cout << " 7.  Modify virtual configuration" << endl;
    cout << " 8.  Destroy Environment" << endl;
    cout << endl;
    cout << "Enter menu item number (any alpha key to exit): ";

    while (true)
    {
        if (!(cin >> in))
        {
            cout << "Invalid entry. Please try again" << endl;
            cin.clear();
            cin.ignore(10000, '\n');
            return 0;
        };
        switch (in)
        {
            case 0:
                break;
            case 1:
                system("cls");
                eidNamePair = selectEnvFromTree();
                system("cls");
                break;
            case 2:
                createVirtuozzoVE();
                break;
            case 3:
                startVirtuozzoVE(eidNamePair.first);
                break;
            case 4:
                stopVirtuozzoVE(eidNamePair.first);
                break;
            case 5:
                restartVirtuozzoVE(eidNamePair.first);
                break;
            case 6:
                system("cls");
                displayEnvInfo(eidNamePair.first);
                system("pause");
                break;
        }
    }
}

```

```

        case 7:
            modifyVirtualConfig(eidNamePair.first);
            break;
        case 8:
            destroyVirtuozzoVE(eidNamePair.first);
            break;
        default:
            cout << "Invalid entry. Please try again" << endl;
            continue;
    }
    break;
}
}
return 0;
}

/*****

// Here we have two functions that build the Environment tree.
// One function will actually build and display the
// tree (it will be called recursively).
// The second function will initially call the first
// function and will allow the user to select
// an Environment from the tree.

// Builds and displays the Environment tree.
// Parameters
// envList :   The Environment list retrieved from
//             the Access Point cache.
// parentEID : Parent Environment ID (the top level
//             of this branch).
// eids :      The list of the retrieved EID/Name pairs.
// indent :   An indentation at which the Environment name
//             should be displayed.

int displayEnvTree(VZLAccessPointPrototype::EnvValuesListSP
                  envList, eid_t parentEID,
                  EIDNameList& eids, int indent)
{
    // The Environment container iterator.
    // The iterator's member access operator ->() returns
    // a pointer to the VZLEnv object, containing
    // an individual Environment information.
    VZLEnvCache::EnvValuesList::const_iterator cit;

    // Regular configuration structure.
    VZLEnvConfig envConfig;

    // Virtuozzo virtual config structure.
    VZAEnvConfig vzEnvConfig;

    // List of the Environment IP addresses.
    VZLEnvConfig::IPAddressList IPList;

    // Environment type.
    VZLEnvConfigBasic::type_t envType;

    // Environment name.
    string envName;
    // Hostname

```



```

string envHostname;
// IP address.
string IPaddress;
// Environment title.
string envTitle;
// Environment index (for display)
int envIndex;

// Iterate through the Environment container.
for (cit = envList->begin(); cit != envList->end(); ++cit)
{
    // The function actually builds one tree branch
    // at a time. Each Environment in the list has
    // the parent EID property, which indicates the
    // parent Environment EID. You can use this property
    // to determine whether an Environment belongs to
    // the current branch. If does, add it to the branch.
    if (cit->getParentEID() == parentEID)
    {
        // some tree formatting...
        cout << string(indent, ' ') << "|" << endl;
        cout << string(indent, ' ') << "|-- ";

        // Based on the Environment type, use the
        // correct configuration structure and then
        // read the Environment configuration info
        // from it.
        envType = cit->getType();

        // Generic environment.
        if (envType == "generic")
        {
            envConfig = cit->getConfig();
            envName = envConfig.getName().c_str();
            envHostname = envConfig.getHostname().c_str();
            envConfig.getAddresses(IPList);
            if (IPList.isSpecified() && IPList->size() != 0)
            {
                IPaddress = IPList->begin()->ip;
            }
        }
        // Virtuozzo Virtual Environment.
        else if (envType == "virtuozzo")
        {
            vzEnvConfig = cit->getVirtualConfig();
            envName = vzEnvConfig.getName().c_str();
            envHostname = vzEnvConfig.getHostname().c_str();
            vzEnvConfig.getAddresses(IPList);
            if (IPList.isSpecified() && IPList->size() != 0)
            {
                IPaddress = IPList->begin()->ip;
            }
        }
        else
        {
            // This is a placeholder for future types.
            // If you have other Environment types,
            // use the correct config structure here.
        }
    }
}

```

```

// There are several fields that can be used
// as the Environment "name". None of those
// fields are mandatory, however. Let's check
// each field and use one that has a value as
// the Environment "Title".
envTitle = envName;
if (envTitle.empty())
{
    // Hostname
    envTitle = envHostname;
    if (envTitle.empty())
    {
        // IP address
        envTitle = IPAddress;
        if (envTitle.empty())
            envTitle = "N/A";
    }
}

// Insert the current EID/Name pair into
// the list. The list will be used later
// to determine the Environment title based on
// the ID (for display purposes).
EIDNamePair pair;
pair.first = cit->getEID();
pair.second = envTitle;
eids.push_back(pair);

// Display the basic Environment properties
// on the screen.
// Title:
envIndex = int(eids.size()) - 1;
cout << "(" << envIndex << ") " << envTitle;

// some screen output formatting...
char buffer[30];
itoa(envIndex, buffer, 10);
cout << string((33 - (strlen(buffer) +
envTitle.length()))), ' ');

// Environment type:
cout << envType.c_str() << "\t";

// Environment status.
cout << getEnvStatusDesc(cit->getStatus()) << endl;

// Go to the next level of recursion to
// build the next tree level.
indent = indent + 3;
displayEnvTree(envList, cit->getEID(), eids, indent);
indent = indent - 3;
}
}
return 0;
}

// The second function (works together with
// displayEnvTree() listed above)
// Allows to select an Environment from the tree.
EIDNamePair selectEnvFromTree()

```

```

{
    // Get the Environment list from Access Point cache.
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();

    // Display heading.
    cout << "Environments\t\t\t\t\tType\t\tStatus" << endl;
    cout << string(65, '-') << endl;

    // Get the root Environment information.
    VZLEnvCSP env = envList->getEnv(accessPoint->getRootEID());

    // A list of EIDs.
    EIDNameList eids;
    // The EID/Name pair.
    EIDNamePair eidNamePair;
    // Populate the pair with root Environment values.
    eidNamePair.first = accessPoint->getRootEID();
    eidNamePair.second = "root";
    eids.push_back(eidNamePair);
    cout << "(0) root - " <<
        env->getConfig().getHostname().c_str() << endl;

    // Build and display the Environment tree,
    // starting with the root Environment.
    // The function will be called recursively for each
    // Environment in the list, because each Environment
    // may have child Environments of its own.
    displayEnvTree(envList, accessPoint->getRootEID(), eids, 0);

    // Allow the user to select an environment from
    // the list. The user must select the Environment
    // index number (a sequential number added to
    // each Environment entry by the displayEnvTree() function).
    // The Environment EID/Name pair is the element of the
    // eids array with the selected index.
    unsigned int iIn;
    while (true)
    {
        cout << endl;
        cout << "Enter Environment index number: ";
        cin >> iIn;

        if (iIn >= eids.size() || iIn < 0)
        {
            cout << "Invalid entry, please try again." << endl;
            iIn = 0;
            continue;
        }

        return eids[iIn];
    }
}

/*****
// A helper function to convert the Environment state and
// transition codes into a description that can be displayed
// to the user. The Environment state and transition are
// enumerations defined in the VZAgent client library.
string getEnvStatusDesc(const VZLEnvStatus& envStatus)

```

```
{
    string statusDesc;

    // Determining the VE state and transition.
    if (envStatus.transition == ENV_TRANS_NONE)
    {
        // VE is not in transition. Display status.
        switch (envStatus.state)
        {
            case ENV_STAT_UNEXIST:
                statusDesc = "dropped";
                break;
            case ENV_STAT_DOWN:
                statusDesc = "stopped";
                break;
            case ENV_STAT_MOUNTED:
                statusDesc = "mounted";
                break;
            case ENV_STAT_SUSPENDED:
                statusDesc = "suspended";
                break;
            case ENV_STAT_RUNNING:
                statusDesc = "running";
                break;
            default:
                statusDesc = envStatus.state;
                break;
        }
    }
    else
    {
        // VE is in transition (its stable state is changing).
        switch (envStatus.transition)
        {
            case ENV_TRANS_CREATING:
                statusDesc = "creating ...";
                break;
            case ENV_TRANS_MOUNTING:
                statusDesc = "mounting ...";
                break;
            case ENV_TRANS_STARTING:
                statusDesc = "starting ...";
                break;
            case ENV_TRANS_STOPPING:
                statusDesc = "stopping ...";
                break;
            case ENV_TRANS_DESTROYING:
                statusDesc = "destroying ...";
                break;
            case ENV_TRANS_BACKING_UP:
                statusDesc = "backing up ...";
                break;
            case ENV_TRANS_RESTOREING:
                statusDesc = "restoring ...";
                break;
            case ENV_TRANS_SUSPENDING:
                statusDesc = "suspending ...";
                break;
            case ENV_TRANS_RESUMING:
                statusDesc = "resuming ...";
                break;
        }
    }
}
```

```

        break;
    default:
        statusDesc = envStatus.transition;
    }
}

return statusDesc;
}

/*****
// Creates new Virtuozzo VE.

int createVirtuozzoVE()
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAF functionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "Error: unable to create VZA::VZAEnvMSP object." <<
endl;
        return -1;
    }

    // Environment ID.
    eid_t envIDout;

    // Environment configuration info.
    VZAEnvConfig envConf;

    int iIn;
    string sIn;

    // VEID
    cout << "VEID: ";
    cin >> iIn;
    envConf.setVeid(iIn);

    // Environment NAME
    cout << "Environment Name: ";
    cin >> sIn;
    envConf.setName(sIn);

    // IP address
    cout << "IP address: ";
    cin >> sIn;
    VZLEnvConfig::IPaddressList::value_type IPlist;
    VZLIPaddress IPaddress;
    IPaddress.ip = sIn;
    IPlist.insert(IPaddress);
    envConf.setAddresses(IPlist);

    // Hostname

```

```

cout << "Hostname: ";
cin >> sIn;
envConf.setHostname(sIn);

// Environment description.
cout << "Description: ";
cin >> sIn;
envConf.setDescription(sIn);

// OS template.
// Frist, get the list of the installed
// standard OS templates.

cout << endl << "Available OS templates:" << endl;

VZA::VZATEMSP vzatem = VZALibFunctionality::kit().getVZATEM(
    accessPoint->getRootEID(),
    accessPoint,
    VZA::vzaFunctionalType);

if (!vzatem)
{
    cerr << "Error: unable to create VZATEMSP object." << endl;
    return -1;
}

VZAPackageInfoList pkgInfoList;
eid_t eid;

// Passing empty eid, which means that
// we want the templates installed in the
// root Environment.
if (vzatem->ls(eid, &pkgInfoList, "os") != 0)
{
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error (" << errorData.code << ") ";
    cout << errorData.message << endl << endl;
}

bool bPkgListEmpty = true;
std::vector<string> pkgList;
int ctr = 0;

if (pkgInfoList.size() != 0)
{
    for (VZAPackageInfoList::iterator pit = pkgInfoList.begin();
        pit != pkgInfoList.end(); ++pit)
    {
        pkgList.push_back(pit->name);
        cout << "(" << ctr++ << ") ";
        cout << pit->name << "\tstandard template" << endl;
    }
}

// Now, get the list of the installed
// EZ templates.
VZA::VZAPkgMSP vzapkgm = dynamic_pointer_cast<VZAPkgM>(
    VZLLibFunctionality::kit().getPkgM(
        accessPoint->getRootEID(),

```

```

        accessPoint,
        VZA::vzaFunctionalType));

if (!vzapkgm)
{
    cerr << "Error: unable to create VZAPkgMSP object." << endl;
    return -1;
}

VZLPackageList pkgOut;
vector<string> pkgType;
pkgType.push_back("os");
VZLPackageManagerListOptions pkgOptions;
pkgOptions.types = pkgType;

vzapkgm->list(accessPoint->getRootEID(),
             &pkgOut, NULL, pkgOptions);

if (pkgOut.size() == 0 && ctr == 0)
{
    cout << "No OS templates were found." << endl;
    system("pause");
    return -1;
}

for (VZLPackageList::iterator pit = pkgOut.begin();
     pit != pkgOut.end(); ++pit)
{
    pkgList.push_back((*pit)->name);
    cout << "(" << ctr++ << " ) ";
    cout << (*pit)->name << "\tEZ Template" << endl;
}
cout << endl;

cout << "Enter template number: ";

cin >> iIn;

envConf.setOsTemplate(VZATemplate(pkgList[iIn]));

// Sample configuration.
initVZLFunctionalityAgent();
VZLEnvSampleMSP vzlSampleConf =
    VZLLibFunctionality::kit().getEnvSampleM(
        accessPoint->getRootEID(),
        accessPoint);

if (!vzlSampleConf)
{
    cerr << "Error: unable to create VZLEnvSampleMBaseSP object."
<< endl;
    return -1;
}

VZLSampleID sampleConfigID;
VZLSampleConfList confs;
VZLSampleIDList ids;
vzlSampleConf->getSampleConf(&confs, ids);

pair<VZLGUID, string> IdNamePair;

```

```

vector<pair<VZLGUID, string> > sampleList;
cout << endl << "Sample configuration list" << endl << endl;

ctr = 0;
for (VZLSampleConfList::iterator sit = confs.begin();
    sit != confs.end(); ++sit)
{
    IdNamePair.first = sit->id;
    IdNamePair.second = sit->name;
    sampleList.push_back(IdNamePair);
    cout << "(" << ctr++ << " ) ";
    cout << sit->name << endl;
}

cout << "Enter sample number: ";
cin >> iIn;
envConf.setBaseSampleId(sampleList[iIn].first);

// Create VE.
cout << endl << "Creating VE ...";
if (vzaenvm->create(&envIDout, envConf) != 0)
{
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error ( " << errorData.code << " ) ";
    cout << errorData.message << endl;
    return -1;
};
cout << endl << endl << "Virtual Environment was created
successfully." << endl;
system("pause");
return 0;
}
/*****
// Displays the specified Environment information.
int displayEnvInfo(eid_t eid)
{
    // Get the Environment list from cache.
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();

    // Get the specified Environment information.
    VZLEnvCSP env = envList->getEnv(eid);

    // Get the VZLEnvConfig object containing the specified
    // Environment configuration info.
    VZLEnvConfig envConfig = env->getConfig();

    // Call the appropriate function to extract and display the
    // configuration info on the screen.
    if (env->getType() == "generic")
    {
        // Displays the generic Environment info.
        displayGenericEnvInfo(eid);
    }
    else if (env->getType() == "virtuozzo")
    {
        // Displays Virtuozzo VE info.
        displayVirtuozzoVEinfo(eid);
    }
}

```



```

else
{
    // This is a placeholder for future types.
    cout << "Unsupported Environment type." << endl;
    return -1;
}

return 0;
}
/*****
// Displays generic Environment information.
int displayGenericEnvInfo(eid_t eid)
{
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();
    VZLEnvCSP env = envList->getEnv(eid);
    VZLEnvConfig envConfig = env->getConfig();

    cout << endl << "Environment Information" << endl << endl;
    cout << "EID:\t\t" << eid.toString() << endl << endl;
    cout << "Name: \t\t" << envConfig.getName().c_str() << endl;
    cout << "Hostname:\t" << envConfig.getHostname().c_str() << endl;
    cout << "Domain:\t" << envConfig.getDomain().c_str() << endl;
    cout << "Type:\t\t" << env->getType().c_str() << endl;
    cout << "Status:\t\t" << getEnvStatusDesc(env->getStatus()) <<
endl;
    cout << "Architecture:\t" << envConfig.getArchitecture().c_str()
<< endl;
    cout << "Description:\t" << envConfig.getDescription().c_str() <<
endl;

    cout << endl << "Operating System:" << endl;
    cout << "    Name: " << envConfig.getOS().name.c_str() << endl;
    cout << "    Platform: " << envConfig.getOS().platform.c_str() <<
endl;
    cout << "    Version: " << envConfig.getOS().version.c_str() <<
endl;
    cout << "    Kernel: " << envConfig.getOS().kernel.c_str() << endl;

    VZLEnvConfig::IPAddressList IPList;
    envConfig.getAddresses(IPList);

    cout << endl << "IP Address:\t";
    for (VZLEnvConfig::IPAddressList::value_type::iterator cit =
        IPList->begin(); cit != IPList->end(); ++cit)
    {
        cout << cit->ip << " " << cit->netmask << " ";
    }
    cout << endl;
    return 0;
};

/*****
// Displays Virtuozzo VE information.
int displayVirtuozzoVEinfo(eid_t eid)
{
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();

```

```

VZLEnvCSP env = envList->getEnv(eid);
VZAEnvConfig envConfig = env->getVirtualConfig();

cout << endl << "Environment Information" << endl << endl;
cout << "EID:\t\t" << eid.toString() << endl << endl;
cout << "Name: \t\t" << envConfig.getName().c_str() << endl;
cout << "Hostname:\t" << envConfig.getHostname().c_str() << endl;
cout << "Domain:\t" << envConfig.getDomain().c_str() << endl;
cout << "Type:\t\t" << env->getType().c_str() << endl;
cout << "Status:\t\t" << getEnvStatusDesc(env->getStatus()) <<
endl;
    cout << "Architecture:\t" << envConfig.getArchitecture().c_str()
<< endl;
    cout << "Description:\t" << envConfig.getDescription().c_str() <<
endl;

    cout << endl << "Operating System:" << endl;
    cout << "    Name: " << envConfig.getOS().name.c_str() << endl;
    cout << "    Platform: " << envConfig.getOS().platform.c_str() <<
endl;
    cout << "    Version: " << envConfig.getOS().version.c_str() <<
endl;
    cout << "    Kernel: " << envConfig.getOS().kernel.c_str() << endl;

VZLEnvConfig::IPAddressList IPList;
envConfig.getAddresses(IPList);

cout << endl << "IP Address:\t";
for (VZLEnvConfig::IPAddressList::value_type::iterator cit =
    IPList->begin(); cit != IPList->end(); ++cit)
{
    cout << cit->ip << " " << cit->netmask << " ";
}
cout << endl;

// Virtuozzo-specific settings
veid_t veid;
envConfig.getVeid(veid);
cout << "VEID:          " << veid;
if (veid == 1)
{
    cout << " (Service VE)";
}
cout << endl;

VZLBoolOptionalProperty bOn;
envConfig.getOnBoot(bOn);
cout << "On-boot:          " << (bOn ? "on" : "off") << endl;

envConfig.getOfflineManagement(bOn);
cout << "Offline mgmt:    " << (bOn ? "on" : "off") << endl;

string rootDir;
envConfig.getVERoot(rootDir);
cout << "Root directory: " << rootDir << endl;

VZAEnvConfig::NetVEthListAttribute netVEthList;
envConfig.getNetVEths(netVEthList);
VZAEnvConfig::NetVEthListAttribute::value_type::iterator iter =
netVEthList->begin();

```

```

    cout << "Netowrk devices:" << endl;
    for (iter; iter != netVEthList->end(); ++iter)
    {
        cout << "    " << iter->toString() << endl;
    }

    cout << endl;
    return 0;
}

/*****
// Starts the specified Virtuozzo VE.
int startVirtuozzoVE(eid_t eid)
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAF functionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "Error: unable to create VZA::VZAEnvMSP object." <<
endl;
        return -1;
    }

    cout << "Starting VE..." << endl;
    if (vzaenvm->start(eid) != 0)
    {
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error ( " << errorData.code << " ) ";
        cout << errorData.message << endl;
        return -1;
    }

    cout << endl << "VE started successfully." << endl;
    return 0;
};

/*****
// Stops the specified Virtuozzo VE.
int stopVirtuozzoVE(eid_t eid)
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAF functionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,

```

```

        VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "Error: unable to create VZA::VZAEnvMSP object." <<
endl;
        return -1;
    }

    cout << "Stopping VE..." << endl;
    if (vzaenvm->stop(eid) != 0)
    {
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error (" << errorData.code << ") ";
        cout << errorData.message << endl;
        return -1;
    }

    cout << endl << "VE has been stopped." << endl;
    return 0;
};

/*****
// Restarts the specifies Virtuozzo VE.

int restartVirtuozzoVE(eid_t eid)
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAF functionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "Error: unable to create VZA::VZAEnvMSP object." <<
endl;
        return -1;
    }

    cout << "Restarting VE..." << endl;
    if (vzaenvm->restart(eid) != 0)
    {
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error (" << errorData.code << ") ";
        cout << errorData.message << endl;
        return -1;
    }

    cout << "VE restarted successfully." << endl;
    return 0;
};

*****/

```

```

// Destroys the specified Virtuozzo VE.

int destroyVirtuozzoVE(eid_t eid)
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAFunctionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "Error: unable to create VZA::VZAEnvMSP object." <<
endl;
        return -1;
    }
    if (vzaenvm->destroy(eid) != 0)
    {
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error ( " << errorData.code << " ) ";
        cout << errorData.message << endl;
        return -1;
    }

    cout << endl << "VE has been destroyed." << endl;
    return 0;
}

/*****/
// Modifies Virtual Environment configuration.

int modifyVirtualConfig(eid_t eid)
{
    // Get the Environment info from the Access Point cache.
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();
    VZLEnvCSP env = envList->getEnv(eid);

    // Virtuozzo VE.
    if (env->getType() == "virtuozzo")
    {
        modifyVirtuozzoVEConfig(env);
    }
    // Normally, there's no virtual configuration for
    // generic Environments.
    else if (env->getType() == "generic")
    {
        cout << endl;
        cout << "Error: not a virtual Environment, unable to
continue.";
        cout << endl;
        return -1;
    }
    // Some other, currently unsupported virtualization technology.
    else

```

```

    {
        // This is a placeholder for future types.
        cout << "Unsupported virtual Environment type." << endl;
    }

    return 0;
}

/*****
// Modifies Virtuozzo VE virtual configuration.

int modifyVirtuozzoVEConfig(VZLEnvCSP env)
{
    eid_t eid = env->getEID();
    VZAEEnvConfig envConfigOld = env->getVirtualConfig();
    VZAEEnvConfig envConfigNew;

    string sIn;
    cout << endl << "Modify configuration" << endl << endl;
    cout << "Current name: " << envConfigOld.getName().c_str() <<
endl;
    cout << "New Name: ";
    cin >> sIn;
    envConfigNew.setName(sIn);

    cout << endl;
    cout << "Current hostname: " << envConfigOld.getHostname().c_str()
<< endl;
    cout << "New Hostname: ";
    cin >> sIn;
    envConfigNew.setHostname(sIn);
    cout << endl;

    VZLBoolOptionalProperty bOn;
    envConfigOld.getOnBoot(bOn);
    cout << "On-boot current: " << (bOn ? "on" : "off") << endl;
    cout << "On-boot (0 = off, 1 = on): ";
    cin >> sIn;
    envConfigNew.setOnBoot((sIn == "1" ? true : false));

    envConfigOld.getOfflineManagement(bOn);
    cout << "Offline mgmt current: " << (bOn ? "on" : "off") <<
endl;

    cout << "Offline management (0 = off, 1 = on): ";
    cin >> sIn;
    envConfigNew.setOfflineManagement(sIn == "1" ? true : false);

    // Create the Environment Management object using object factory.
    VZA::VZAEEnvMSP vzaenvm = dynamic_pointer_cast<VZAEEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "Error: unable to create VZA::VZAEEnvMSP object." <<
endl;
        return -1;
    }
}

```

```
}

if (vzaenvm->set(eid, envConfigNew) != 0)
{
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error ( " << errorData.code << " ) ";
    cout << errorData.message << endl;
    return -1;
}

cout << endl;
return 0;
```

Adding Environment Monitoring

```
}
```

This part of the tutorial shows you how to implement the Environment monitoring functionality. Environment monitoring includes the following activities:

- Performance threshold monitoring. You can retrieve the latest performance reports, or you can use subscription services to receive performance report on a periodic basis.
- Monitoring system alerts. You can retrieve a list of all currently raised alerts, or you can use subscription services to receive alert notifications automatically.
- Monitoring an Environment for critical system events.

The following sections describe in detail how to implement each monitoring function in your programs.

1. Retrieving a List of Performance Counters

The Environment performance thresholds are tracked using special components called *performance counters*. Performance counters contain the values that can be used to analyze the system performance, identify the bottlenecks, and then use this information to fine-tune the system.

Performance counters are organized into *performance classes*. Each performance class contains the counters that capture the information about a particular type of system resource (e.g. CPU, memory, disk, network). Each counter within a class contains a value describing a specific resource activity. For example, the "network interface" class has the "incoming traffic" and "outgoing traffic" counters, among others. The "CPU" class has "cpu system", "cpu user", "cpu idle", and other counters.

There are two possible ways that you can work with performance counters. The first approach is to retrieve the performance data for all of them at once. For this, you don't have to know the actual names of the classes and the counters in advance. The second approach is used when you want to get the performance data for a particular system resource. In this case, you will have to know the exact name of the class and the names of its counters. The list of the available classes and the counters is stored in the VZAgent vocabulary. The rest of this section explains how to retrieve them.

Before we continue, let's talk about one more component of the performance data storage structure. This component is *class instance*. When you have more than one device of the same type, each device (from the performance data collection point of view) is an instance of a class. For example, if you have more than one network interface, each interface is an instance of the "network interface" class. If you have more than one hard drive, each drive is an instance of the "disk" class, and so forth. If you have more than one instance of a class, you have to specify the instance for which you want the information. If you don't specify the instance name, and there's more than one instance exists, then you will receive the information for all of them. The name of the class instance that you should use is the actual name of the device assigned to it by the operating system.

Now that you know what classes, instances, and counters are, let's begin writing the code that will retrieve the names of the classes and the counters. As we said earlier, the names are stored in the VZAgent vocabulary. If you want to monitor the Environment that you are connected to (i.e the root Environment), the vocabulary located on that machine must be used. If you want to monitor a virtual Environment, such as Virtuozzo VE, use the vocabulary located on the Environment hosting that virtual Environment (in many cases this is also going to be the root Environment, that is if the VE is hosted by it).

Add the following `#include` directive to your program:

```
#include <VZLFunctionality/VZLPerfMon.h>
```

Declare a new function that will retrieve the names of the classes and the counters:

```
int selectPerformanceCounters(string &classOut, string &counterOut);
```

Let's now write the function. First, get the entire vocabulary from the root Environment.


```
VZLVocMapSP vocabulary = accessPoint->getVocabulary(accessPoint-
>getRootEID());
```

To retrieve the information that you need (classes and counters), you have to create the vocabulary category filter, which can be used to retrieve the information from the vocabulary selectively.

```
// Vocabulary category filter.
VZLCategoryRestrictSet categories;
```

The filter parameters are specified using the `VZLCategoryRestrictSet::insert` method. The first filter parameter is set to retrieve only the information pertinent to Virtuozzo VEs. To retrieve the counters for other Environment types, use the appropriate type name. To retrieve the counters for all types, simply do not "insert" anything here.

```
categories.insert(vzaFunctionalType.c_str());
```

The next filter parameter is set to retrieve only the performance counters categories (these will be actually the names of the performance classes).

```
categories.insert("counters");
```

Use the iterator to retrieve the classes from the vocabulary.

```
auto_ptr<VZLVocMapIterator> vocIter(vocabulary-
>readCategory(categories));

// Check that vocabulary actually contains the
// performance classes categories.
if (vocIter.get() == NULL || vocIter->firstCategory() != 0)
{
    return -1;
}
```

Display the retrieved classes on the screen.

```
int ctr = 0;
string className;

// The classes will be stored in this container.
// The vector index will be used as class ID to allow
// easy selection of the class by the user of the program.
std::vector<string> classIDList;

cout << "Performance Classes:" << endl << endl;
do
{
    vocIter->getID(className);
    classIDList.push_back(className);
    cout << "(" << ctr << " ) " << className << endl;
    ctr++;
} while (vocIter->nextCategory() == 0);
```

Allow the user to select a class from the list.

```
int classID;
cout << endl << "Enter class ID: ";
if (!(cin >> classID))
{
    cin.clear();
    cin.ignore(10000, '\n');
    cout << "Invalid class ID." << endl;
    return -1;
}
```

```

}
if (classID > int(classIDList.size()) || classID < 0)
{
    cout << "Invalid class ID." << endl;
    return -1;
}
cout << endl;

```

Now that you have the name of the class, you can retrieve the names of the counters.

```

auto_ptr<VZLVocMapIterator> cntrIter(vocabulary-
>readCategory(classIDList[classID]));

// Get the counters and check that vocabulary contains
// counters for the specified class.
if (cntrIter.get() == NULL || cntrIter->firstParameter() != 0)
{
    cout << endl;
    cout << "No counters are available for the specified class.";
    cout << endl;
    return -1;
}

// A vector to store the names of the counters.
// Vector index will correspond to the
// counter ID displayed on the screen to allow
// an easy selection of the counter by the user.
std::vector<string> counterIDList;
ctr = 0;

// Get the list of counters for the selected class.
cout << "Counters: " << endl << endl;
do
{
    string counterName;
    int valueType, counterType;

    // Get counter information.
    cntrIter->getID(counterName);
    cntrIter->getValue(valueType, "value_type");
    cntrIter->getValue(counterType, "counter_type");

    // Insert the counter name into the "ID" vector.
    // Display counter information on the screen.
    counterIDList.push_back(counterName);
    cout << "(" << ctr << " ) " << counterName << " ( " <<
        ("valueType == 1" ? "integer" : "float") << ", " <<
        ("counterType == 1" ? "incremental" : "absolute") <<
        ")" << endl;
    ctr++;
} while (cntrIter->nextParameter() == 0);

```

Allow the user to select a counter from the list (this can be actually modified to allow the selection of multiple counters if needed).

```

int counterID;
cout << endl << "Enter counter ID: ";

if (!(cin >> counterID))
{
    cout << "Invalid counter ID." << endl;
    return -1;
}

```

```

}

if (counterID > int(counterIDList.size()) || counterID < 0)
{
    cin.clear();
    cin.ignore(10000, '\n');
    cout << "Invalid counter ID." << endl;
    return -1;
}

cout << endl;

```

The function parameters used as output are assigned the values of the selected class and the counter, and the function returns.

```

classOut = classIDList[classID];
counterOut = counterIDList[counterID];

return 0;

```

When this function is executed, the output should look similar to the following example:

```

Performance Classes:

(0) counters_cpu
(1) counters_vz_ubic
(2) counters_net
(3) counters_vz_quota
(4) counters_loadavg
(5) counters_process
(6) counters_system
(7) counters_vz_slm

Enter class ID: 0

Counters:

(0) counter_cpu_system (integer, incremental)
(1) counter_cpu_user (integer, incremental)
(2) counter_cpu_idle (integer, incremental)
(3) counter_cpu_nice (integer, incremental)
(4) counter_cpu_vz_starvation (integer, incremental)
(5) counter_cpu_system_states (integer, incremental)
(6) counter_cpu_user_states (integer, incremental)
(7) counter_cpu_idle_states (integer, incremental)
(8) counter_cpu_nice_states (integer, incremental)
(9) counter_cpu_vz_starvation_states (integer, incremental)
(10) counter_cpu_system (integer, incremental)
(11) counter_cpu_user (integer, incremental)
(12) counter_cpu_nice (integer, incremental)
(13) counter_cpu_idle (integer, incremental)
(14) counter_cpu_system_states (integer, incremental)
(15) counter_cpu_user_states (integer, incremental)
(16) counter_cpu_nice_states (integer, incremental)
(17) counter_cpu_idle_states (integer, incremental)

Enter counter ID: _

```

The information in parenthesis indicates the data type of the counter's value (e.g. integer), and the type of the counter. There are two types of counters: incremental and absolute. The current value of an incremental counter represents the total number of units since the Environment was started (for example, the total incoming network traffic in bytes). Absolute counters contain the last recorded value.

Now that you have selected the class and the counter, you can get the performance reports for this particular counter (instead of retrieving it for all available counters). The following sections will show you how to get the actual reports.

2. Getting a Single Performance Report

Let's declare another function.

```
int getPerformanceReport(eid_t eid);
```

The function will retrieve a single performance report, which will display the latest available values for the selected counters.

First, use the function that you created in the previous section to get the list of classes and the counters.

```
string className;
string counterName;

if (selectPerformanceCounters(className, counterName) != 0)
{
    return -1;
}
```

The function call above will display the list of classes and will allow the user to select a class from the list. It will then display the list of counters and will allow the user to select a counter from the list. You can now use the returned class and counter names to get the performance data.

Store the selected classes and counters in the container. This will be passed later to the method that will retrieve the performance data. If there are multiple instances of the class in your system (e.g. multiple network interfaces), the actual name of the instance must be used instead of the empty string. To retrieve the data for multiple counters, the name of each counter must be inserted into the container in the same exact manner as shown in the example below.

```
VZLPerfMonClassListSP classList(new VZLPerfMonClassList);
std::string instanceName = "";
(*classList)[className][instanceName].insert(counterName);
```

The next step is to create the VZLPerfMon object (performance monitor) using the object factory.

```
// Initialize VZL plugin.
initVZLFunctionalityAgent();

// Create VZLPerfMon object.
VZLPerfMonSP perfmon;
perfmon = VZLLibFunctionality::kit().getPerfMon(
    accessPoint->getRootEID(),
    accessPoint);
```

You now have a choice of retrieving the data for all Environments known to VZAgent, or for a specific Environment. To retrieve the data for a specific Environment, do the following.

```
VZLEIDList eidList;
eidList.insert(eid);
```

The eidList container will be passed to the method that retrieves the performance data in the next step. To retrieve the data for all Environments, leave the container empty.

Finally, get the performance data. On method return, the `monListOut` object will contain the requested data.

```
VZLMonitorDataList monListOut;
perfmon->get(&monListOut, eidList, classList, false);
```

The boolean parameter (the last one) controls how incremental counters are handled. If set to true, the difference between the last two values will be reported. Otherwise, the current value of a counter will be returned.

The `monListOut` object should now contain the requested performance data. Iterate through it and display the data on the screen. If multiple classes were selected, each iteration displays the data for an individual class.

```
for(VZLMonitorDataList::const_iterator it =
    monListOut.begin();
    it != monListOut.end(); ++it)
{
    const VZLMonitorData& data = *it;

    cout << "Class:\t" << data.counterClass << endl;
    cout << "From:\t" << ctime(&data.interval.startTime);
    cout << "To:\t" << ctime(&data.interval.endTime);
    cout << endl;

    // This loop displays the data for each
    // class instance (if more than one instance exists).
    for(VZLMonitorData::InstanceMap::const_iterator
        inst = data.counters.begin();
        inst != data.counters.end(); ++inst)
    {
        cout << "Instance: " <<
            ("inst->first.length() == 0" ? "N/A" : inst->first)
            << endl << endl;

        // This loop displays the information for each
        // specified counter.
        for(VZLMonitorData::CountersMap::const_iterator counter =
            inst->second.begin();
            counter != inst->second.end(); ++counter)
        {
            cout << "Counter: " << counter->first << endl << endl;
            cout << "min\tmax\tavg\tcur" << endl;
            cout << string(30, '-') << endl;
            cout << counter->second.min.intval << "\t";
            cout << counter->second.max.intval << "\t";
            cout << counter->second.avg.intval << "\t";
            cout << counter->second.cur.intval << endl << endl;
        }
    }
    cout << endl;
}
return 0;
```

And the output should look similar to the following example.

```
Class:   counters_cpu
From:    Wed Oct 18 06:38:48 2006
To:      Wed Oct 18 06:38:58 2006

Instance: N/A
```

```
Counter: counter_cpu_system
```

min	max	avg	cur
1	1	1	10933

3. Receiving Periodic Performance Reports

So far, we've used only the on-demand VZAgent requests. A single on-demand request can produce a single response only (or none at all if the request is procedural in nature). In other words, the server will generate a single set of data or execute a single procedure for a given on-demand request. In the previous section, you've learned how to generate a system status report using the on-demand VZAgent functionality. But how do you monitor the system on a periodic basis? You can execute an on-demand request in the main program loop using some time counter of course, but there's a better way to accomplish this goal -- you can use *periodic* requests. Here's how it works.

Periodic requests are executed asynchronously and are implemented using multithreading. During an asynchronous operation, you initiate the request in the main thread, but the request itself is processed in a different thread ("in a background") without interrupting the main program flow. A pointer to a special object called *handler* is passed to the method processing the request. When an event of one of the predefined types triggers in the background thread, it interrupts the main program flow and automatically invokes a method in a handler object that "receives the event". The method can then handle the event. Once the event processing is completed, the control is returned to the main program. Consider the following example that demonstrates how this works with performance monitoring.

- 1 You execute the initial "start monitor" request specifying the performance counters that you would like to monitor. You also pass a pointer to the handler object, which will be receiving the responses from the server side. The "start monitor" call immediately returns without waiting for the server side to respond. Your main program continues executing normally.
- 2 The performance reports are generated automatically at the predefined time intervals on the server side. As soon as the report is ready, the server sends it to your client program. The background thread receives the data (transparently to your main program) and then invokes the handler's `processData` method passing the received performance data to it. The main thread yields to the background thread at this point. The code that you put into the method's body can then handle the received data -- it can display the data on the screen, or log it to a file, or do whatever you require.
- 3 As soon as the code in the `processData` method completes processing the data, the control is returned to the main thread and your program continues executing normally again.
- 4 Once another data set is ready on the server side, the `processData` is invoked again, and the cycle repeats. This continues for as long as your connection stays active or until you stop the monitor.

Before you can use this functionality, you have to implement your own handler class. VZAgent C++ library has a number of handler prototype classes, each tailored to handle a particular type of operation. The base prototype for performance monitor handler is VZLMonitorDataHandlerPrototype class. Here's how you implement the handler.

#includes directive:

```
#include <VZLFunctionalityAgent/VZLMonitorAgent.h>
```

Handler class definition:

```
class PerfMonHandler : public VZLMonitorDataHandlerPrototype
{
public:
    void PerfMonHandler::handleOk()
    {
        // Invoked on Monitor start and finish.
    }

    // Invoked on error.
    void handleError(const VZLRequestErrorData &err)
    {
        cout << endl;
        cout << err.code << ", " << err.message.c_str();
        cout << endl;
    }

    // Invoked when the performance data is received.
    void processData(const VZLMonitorDataList& datalist)
    {
        // Get the data from the container
        // and display it on the screen.
        for(VZLMonitorDataList::const_iterator it =
            datalist.begin(); it != datalist.end(); ++it)
        {
            const VZLMonitorData& data = *it;

            // First, display the environment ID,
            // the class name, the beginning and the end
            // time of the report.
            cout << endl;
            cout << "EID:\t" << data.eid.toString() << endl;
            cout << "Class:\t" << data.counterClass << endl;
            cout << "From:\t" <<
                ctime(&data.interval.startTime);
            cout << "To:\t" << ctime(&data.interval.endTime)
                << endl;
            cout << "\t\tmin\tmax\tavg\tcur" << endl << endl;

            // Now, display the actual counter data.
            for(VZLMonitorData::InstanceMap::const_iterator
                inst = data.counters.begin();
                inst != data.counters.end(); ++inst)
            {
                cout << "Instance:\t" << inst->first << endl;
                for (VZLMonitorData::CountersMap::const_iterator
                    counter = inst->second.begin();
                    counter != inst->second.end();
                    ++counter)
                {
```



```

        cout << "\t\t" << counter->first
              << ":" << endl;
        cout << "\t\t" <<
              counter->second.min.intval << "\t";
        cout << counter->second.max.intval << "\t";
        cout << counter->second.avg.intval << "\t";
        cout << counter->second.cur.intval
              << endl << endl;
    }
    }
    cout << endl;
}
cout << string (65, '=') << endl;
}

};

// A smart pointer must be used to declare the handler
// object variable.
typedef intrusive_smart_pointer<PerfMonHandler> PerfMonHandlerSP;

// Declare and create the handler object.
PerfMonHandlerSP perfMonHandler(new PerfMonHandler());

```

The most important method in the `PerfMonHandler` class above is the `processData` method. Once the monitor is started, the method will be called automatically by the background process every time a performance report is received from the server side. The performance data is passed to the method through the `datalist` parameter. As you can see in the code above, we process the data by examining the `datalist` container and displaying the data on the screen.

Declare the function that will start the monitor. The function accepts the handler object and EID of the Environment to monitor.

```
int startPerfMon(PerfMonHandlerSP perfMonHandler, eid_t eid);
```

Let's now write the code that you'll put into the function body.

Initialize the VZL plugin.

```
initVZLFunctionalityAgent();
```

Create the monitor object. The constructor accepts the EID of the Environment to monitor, and a smart pointer to the access point object.

```
VZLMonitorAgent perfMon(accessPoint->getRootEID(), accessPoint);
```

Select the classes and the counters that you would like to use. For this, we will use the `selectPerformanceCounters` function that you created earlier in this tutorial. Again, in this example we are selecting a single class and a single counter. You may select as many classes and counters as you like, or you may skip the step altogether to monitor all available classes and counters.

```

string className;
string counterName;

if (selectPerformanceCounters(className, counterName) != 0)
{
    return -1;
}

```

```
}
```

Insert the selected class and the counter into the container so they can be passed to the monitor. In case of multiple classes and counters, this code must be repeated for all of them. If you have more than one instance of a class, and you would like to monitor a particular one, use its name instead of an empty string. In our example, all instances of a class (if more than one instance exists) will be monitored.

```
VZLPerfMonClassList classes;

string instanceName = "";
classes[className][instanceName].insert(counterName);

VZLEIDList eidList;
eidList.insert(eid);
```

Finally, start the monitor. The third parameter (the value 10 that we are passing here) sets the reporting period in seconds. You can use any other value if you wish. The last parameter (`VZLPerfMonEnvFilter()`) allows you to specify the type of the Environments to monitor. It only makes sense when you are not populating the `eidList` container (meaning that you want to monitor all available Environments), so by specifying the list of the Environment types, you can limit the monitoring to those types only. In our example, we are specifying the Environment ID, so we pass an empty `VZLPerfMonEnvFilter` object.

```
cout << "Starting monitor..." << endl;
perfMon.async(perfMonHandler)->start(eidList, classes, 10,
    VZLPerfMonEnvFilter());

return 0;
```

When you executed the function, you should see the output on your screen similar to the following example every 10 seconds (the actual values depend on the class and the counter that the user selects):

```
=====
EID:      010783bd-91e8-4a85-b90c-2874b03d8cf6
Class:    counters_net
From:     Thu Oct 19 01:49:46 2006
To:       Thu Oct 19 01:49:47 2006

           min      max      avg      cur
Instance:  eth0
           counter_net_incoming_packets:
           6         6         6      876541

Instance:  lo
           counter_net_incoming_packets:
           0         0         0       104

Instance:  venet0
           counter_net_incoming_packets:
           0         0         0       361
=====
```

To stop the monitor, use the following code snippet:

```
if (perfMonHandler->stop() == 0)
{
    cout << "Monitor has been stopped." << endl;
}
```

4. Receiving System Event Notifications

There are two types of system events that you can monitor from your VZAgent client program (additional events may be added in the future):

- Environment status change -- the status of the Environment may change due to user interaction, system error, license expiration or installation, or possibly some other reason.
- Environment configuration change -- triggers when the user modifies the Environment configuration information.

Event notification messages are received by the client program asynchronously, similarly to the performance monitor functionality. You subscribe to the desired event notification service once and VZAgent automatically notifies you every time an event takes place.

Similar to the performance monitor functionality, event notifications are sent to the client program through handlers, so the first thing that you have to do is implement the handler class. Each subscription type requires its own handler class, which must be derived from the appropriate prototype class. The following example shows how to implement event handlers.

Status change events

```
class StatusEventHandler: public VZLEnvStatusEventReceiver
{
public:

    // Called from the background thread when
    // an event notification is received.
    // The "event" parameter contains the event information.
    void handleEvent(const VZLEnvStatusEvent &event)
    {
        // You may put your own code here.
        // For this demonstration, we simply display the
        // event information on the screen.
        cout << endl;
        cout << string(65, '-') << endl;
        cout << "VE status change detected:" << endl;
        cout << "Subscription name: " << event.subscriptionName <<
endl;
        cout << "Event category: " << event.category << endl;

        VZLInfo info;
        event.toInfo(info);
        cout << info.toString() << endl;

        // The getEnvStatusDesc function determines the Environment
        // status description based on the status code.
        // We implemented the function earlier in this tutorial.
        cout << "Old status: " << getEnvStatusDesc(event.oldStatus) <<
endl;
        cout << "New status: " <<
getEnvStatusDesc(event.newStatus.state) << endl;
        cout << string(65, '-') << endl;
    };
};
```

```
};
```

Let's now write the code that will subscribe to the status change event notification service. First, create the handler object.

```
VZLEnvStatusEventReceiverSP eventHandler(new StatusEventHandler());
```

To subscribe, you must obtain a pointer to the *subscriber* object. You can create the object yourself, but the easiest (and recommended) way to do that is to get the pointer from the Access Point object. Access Point already has a subscriber object because it is also receiving event notifications for its own use.

```
VZLEnvStatusEventSubscriberSP statusSubscriber;  
susbscriber = accessPoint->getEnvStatusEventSubscriber();
```

Finally, you subscribe to the event notification service.

```
if (susbscriber->subscribe(eventHandler, -200) != 0)  
{  
    cout << "Error, unable to subscribe." << endl;  
    return -1;  
}
```

The `-200` argument in the call above specifies the request priority. This value determines the order in which the subscription requests will be processed on the server side. The greater the number, the higher the priority is. This may be important if you want to receive the notifications sooner than the Access Point updates its cache. The Access Point event subscription is using priority `-100` by default. The value `-200` that we use here means that Access Point will update its cache first, and only after that your client program will receive the notification, which means that after you receive the notification, you may safely retrieve the new information from the cache. Please read more about Message Classifications and Priorities in the Programmer's Guide.

After you subscribe to the event notification service, your program continues to run normally. As soon as the status of one of the Environments changes, the server will send a message to your client program containing the event information. Your program will receive the message through handler's `handleEvent` method. To test the code, call the `subscribeEventStatusChange` function, then go and manually stop or start one of the Environments. You should see the `handleEvent` method invoked displaying the event information on the screen similar to the following example.

```
-----  
VE status change detected:  
Subscription name: env_status_subscription  
Event category: env_status  
Environment ae4cb4ae-4c68-4943-ac0c-741246b2c82e status changed  
from 6(0) to 6(5)  
  
Old status: stopped  
New status: running  
-----
```

Please note that the handler's `handleEvent` method may be invoked more than once for a single event. For example, if you are stopping an Environment, the status first changes from "running" to transitional status "stopping" and only then it's changing to the stable status "stopped". Your client program may not receive notifications of the transitional statuses if the Environment stays in that status for a very short period of time, but it will always receive the final status change information.

To cancel the subscription, execute the following code snippet.

```
statusSubscriber->unsubscribe(eventHandler);
```

Environment configuration change events

Configuration change events use their own handler. Here's how you implement it.

```
class ConfigEventHandler: public VZLEnvConfigEventReceiver
{
public:
    // Called from the background thread when
    // an event notification is received.
    // The "event" parameter contains the event information.
    void handleEvent(const VZLEnvConfigEvent &event)
    {
        // You may put your own code here.
        // For this demonstration, we simply display the
        // event information on the screen.
        cout << endl;
        cout << string(65, '-') << endl;
        cout << "VE config change detected:" << endl;
        cout << "Subscription name: " << event.subscriptionName <<
endl;
        cout << "Event category: " << event.category << endl;
        VZLInfo info;
        event.toInfo(info);
        cout << "Message: " << info.toString() << endl;
    };
};
```

The `VZLEnvConfigEvent` object (the `event` parameter of the `handleEvent` method) has two more members. The `config` member contains the regular configuration information. The `vconfig` member contains the virtual configuration information. Physical Environments usually have only the regular configuration. Virtual Environments have both regular and virtual configuration. Virtual configuration information resides outside the virtual Environment, and is used by the virtualization product (e.g. Virtuozzo) to configure the Environment during startup. Depending on your needs, you may examine the values contained in these objects. Please see [Getting Environment Configuration](#) (on page 28) for the examples. For more information on regular and virtual configurations, please see the [Programmer's Guide](#).

Now that you've implemented the handler, you can write the code that will subscribe to the event notification service. First, create the handler object.

```
VZLEnvConfigEventReceiverSP configHandler(new ConfigEventHandler());
```

Obtain a pointer to the subscriber object.

```
VZLEnvConfigEventSubscriberSP configSubscriber;
configSubscriber = accessPoint->getEnvStatusEventSubscriber();
```

Subscribe to the event notification service.

```
if (configSubscriber->subscribe(eventHandler, -200) != 0)
{
    cout << "Error, unable to subscribe." << endl;
}
```

You can test the code by running it and then changing some of the Environment configuration information. You can change the configuration manually on the server side, or you can see the example on how to do it from your client program in the [Modifying Environment Configuration](#) section (on page 36). Shortly after you modify the Environment configuration, you should see the `handleEvent` method invoked displaying the event information on the screen similar to the following example.

```
-----  
VE config change detected:  
Subscription name: env_config_subscription  
Event category: env_config  
Message: Environment ae4cb4ae-4c68-4943-ac0c-741246b2c82e config  
changed.  
-----
```

To cancel the subscription, execute the following code snippet.

```
configSubscriber->unsubscribe(configHandler);
```

Other event types

Future versions of VZAgent software might support additional event types. Third-party plugins might implement their own events as well. Please see the latest VZAgent documentation (Programmer's Guide and Reference) or the plugin documentation for more information.

5. Getting a List of Current Alerts

Alert is a notification describing a circumstance relevant to a healthy system operation. When an alert is raised, it means that there's a problem or an impending problem that may require attention. Alerts are divided into several categories, each handling a particular area of the system (e.g. resource allocation alert, license expiration alert, etc.). The color-coded scheme is used to reflect the severity level of the problem (green, red, yellow, black).

An Environment may have multiple alerts raised at any given time. In this section, you will learn how to retrieve the list of current alerts.

Add an `#include` directive.

```
#include <VZLFunctionality/VZLAlertM.h>
```

Declare a new function.

```
int getAlertList();
```

The following code goes into the function body. First, you have to create the Alert Management object using the object factory.

```
VZL::VZLAlertMSP vzlAlertm;
vzlAlertm = VZLLibFunctionality::kit().getAlertM(
    accessPoint->getRootEID(),
    accessPoint);
```

The `getAlerts` method retrieves the currently raised alerts. The method scans the entire Environment tree starting with the root Environment (the Environment that you are connected to).

```
VZLAlertList pOut;
if (vzlAlertm->getAlerts(&pOut) != 0)
{
    cout << "Error, unable to get alert data." << endl;
    return -1;
}
```

The `getAlerts` method populates the `VZLAlertList` object (`pOut` parameter) with the current alert information (if any). Once the method returns, you may examine the object and extract the alert information that you require. The information includes a text message containing the alert summary info, the ID of the Environment that generated the alert (for virtual Environments, it is usually the parent Environment), the ID of the affected Environment, the alert category, the relevant values, and other information.

Declare an object to store the alert data.

```
VZLAlertDataSP alertData;
```

`VZLAlertList` is a vector of type `VZLAlertSP`. Iterate through it, and extract the information.

```
for (unsigned int i = 0; i < pOut.size(); ++i)
{
```

Getting a message containing the summary alert info.

```
    cout << endl;
    cout << "Message: " << pOut[i]->info.toString() << endl << endl;
```

Get the base alert information. This information is common to all alert categories.

```
cout << "ALERT: " << endl;
cout << "Generated by (EID):\t" << pOut[i]->eid.toString() <<
endl;
cout << "Alert source:\t\t" << pOut[i]->source << endl;
cout << "Alert category:\t\t" << pOut[i]->category << endl;
```

The `VZLAlertList::data` member contains the rest of the alert data. Depending on the alert category, the data member will be an instance of one of the descending classes of the `VZLAlertData` class (the base class that serves as the alert data container). The object, therefore, will always typecast to the `VZLAlertData` class, which will be enough to retrieve some of the most common information.

Typecast the data member to `VZLAlertData` and get the alert severity level.

```
alertData = dynamic_pointer_cast<VZLAlertData>(pOut[i]->data);
cout << "Environment ID:\t\t" << alertData->eid.toString() <<
endl;
cout << "Alert level:\t\t";

switch (alertData->type)
{
    case VZLAlertData::AT_GREEN:
        cout << "Green alert";
        break;
    case VZLAlertData::AT_YELLOW:
        cout << "Yello alert";
        break;
    case VZLAlertData::AT_RED:
        cout << "Red alert";
        break;
    case VZLAlertData::AT_BLACK:
        cout << "Black alert";
        break;
    default:
        cout << "Unknown alert type";
        break;
}
cout << endl;
```

The information specific to a particular alert category is retrieved by typecasting the received object to the appropriate data type. The data type to typecast the object to is determined based on the alert category. At the time of this writing, the only available alert category is `resource_alert` (resource allocation). The name of the class that is used as a container for this alert type is `VZLResourceAlertData`.

```
if (pOut[i]->category == "resource_alert")
{
    VZLResourceAlertDataSP resourceAlertData =
        dynamic_pointer_cast<VZLResourceAlertData>(pOut[i]->data);
```

Get the name of the affected performance counter class.

```
cout << "Counter class:\t\t" << resourceAlertData-
>counterClass << endl;
```


Get the name of the instance of the performance counter class. A class may not have an instance (see [Retrieving a List of Performance Counters](#) (on page 56) for details), so this member is defined as optional. For optional values, you have to check if the value was actually specified, when the container was populated on the server side.

```
        if (resourceAlertData->instance.isSpecified())
        {
            cout << "Instance:\t\t" << resourceAlertData-
>instance.get() << endl;
        }

        cout << "Counter:\t\t" << resourceAlertData->counter << endl;
```

Resource allocation alert may have its values specified as integers or floats. You have to check the actual data type used before reading the values.

```
        // integer value
        if (resourceAlertData->bInteger)
        {
            cout << "Current Value:\t\t" << resourceAlertData-
>cur.intval << endl;
            cout << "Hard Limit:\t\t" << resourceAlertData-
>hard.intval << endl;
            if(resourceAlertData->soft.isSpecified())
            {
                cout << "Soft Limit:\t\t" << resourceAlertData->soft-
>intval << endl;
            }
        }
        // float value
        else
        {
            cout << "Current Value:\t\t" << resourceAlertData-
>cur.floatval << endl;
            cout << "Hard Limit:\t\t" << resourceAlertData-
>hard.floatval << endl;
            if(resourceAlertData->soft.isSpecified())
            {
                cout << "Soft Limit:\t\t" << resourceAlertData->soft-
>floatval << endl;
            }
        }
    }
}
return 0;
// end of the getAlertList function.
```

When you execute the function, the output should look similar to the following example (to raise an alert, you should create a condition on your server that will violate some of resource allocation limits).

Output

```
Message: Resource red physpages alert on environment ae4cb4ae-4c68-
4943-ac0c-741
246b2c82e current value: 2147481420 soft limit: 0 hard limit:
2147483647

ALERT:
Generated by (EID):      010783bd-91e8-4a85-b90c-2874b03d8cf6
Alert source:           resource_alert_monitor
```

```
Alert category:      resource_alert
Environment ID:      ae4cb4ae-4c68-4943-ac0c-741246b2c82e
Alert level:         Red alert
Counter class:       counters_vz_ubc
Counter:             physpages
Current Value:       2147481420
Hard Limit:          2147483647
Soft Limit:          0
```

6. Receiving Alerts by Subscription

In this section you'll learn how to receive alerts automatically using subscription. This functionality works similarly to the performance monitor and event notifications:

- 1** Implement a handler class, which will be used to receive alert data from the server side.
- 2** Subscribe to alerts using an alert subscriber object passing a pointer to the handler object to it.
- 3** When an alert is raised on an Environment, the server will send the alert information to your client program through handler. You examine the alert data and take the appropriate action (e.g. display the alert information on the screen).

The alert handler class must be inherited from the `VZLAlertDataReceiver` class, which is the alert handler prototype class. The following is an example of the alert handler class implementation.

```
class AlertHandler: public VZLAlertDataReceiver
{
public:

    // This method will be automatically called by the
    // background thread as soon as the alert is raised on
    // an Environment, and the alert data is received from
    // the server.
    void handleEvent(const VZLAlertData &data)
    {
        VZLInfo info;
        data.toInfo(info);
        cout << endl;
        cout << string(65, '-') << endl;

        // Display the alert type on the screen.
        switch (data.type)
        {
            case VZLAlertData::AT_GREEN:
                cout << "GREEN ALERT";
                break;
            case VZLAlertData::AT_YELLOW:
                cout << "YELLOW";
                break;
            case VZLAlertData::AT_RED:
                cout << "RED ALERT";
                break;
            case VZLAlertData::AT_BLACK:
                cout << "BLACK";
                break;
            default:
                cout << "ALERT (unknown type)";
                break;
        }

        // Display the rest of the alert information.
        cout << endl << endl;

        // The ID of the affected Environment.
        cout << "EID: " << data.eid.toString() << endl;

        // Subscription name.
        cout << "Subscription: " << data.subscriptionName << endl;

        // A text message describing the alert.
        cout << "Message: " << info.toString() << endl;
        cout << string(65, '-') << endl;
        cout << endl;
    };
};
```

To subscribe to alerts, first obtain a pointer to the alert subscriber object from the Access Point object.

```
if (!alertSubscriber)
{
```

```
    alertSubscriber = accessPoint->getAlertSubscriber();
}
```

Subscribe to alerts. The -200 argument specifies the request processing priority (see [Receiving System Event Notifications](#) (on page 67) for explanation).

```
if (alertSubscriber->subscribe(alertHandler, -200) != 0)
{
    cout << "Error, unable to subscribe to alerts." << endl;
}
```

The subscription covers all Environments in the current Environment tree, including root Environment. To test the code, execute the `subscribeAlert()` function (above) and then create a situation on one of the Environments where some limit (e.g. CPU usage) is exceeded.

When you receive an alert, the function will display an output similar to the following example.

```
-----
RED ALERT

EID: eb521822-caf4-4979-a9c9-4d9947f48dcc
Subscription: alerts_subscription
Message: Resource red physpages alert on environment eb521822-caf4-
4979-a9c9-4d9
947f48dcc current value: 13132 soft limit: 0 hard limit: 2147483647
-----
```

To unsubscribe from alerts, use the following code snippet.

```
alertSubscriber->unsubscribe(alertHandler);
```

CHAPTER 3

Request Redirection

One of the important VZAgent features that you should be aware of is *request redirection*. As the name implies, it allows you to redirect a request to another Environment for processing. To better understand why this feature is useful, consider the following example.

Suppose that you have a physical Windows or Linux machine with VZAgent running on it. Let's say that you also have a server virtualization product, such as Virtuozzo or VMware, running on the same machine. The typical administration tasks that you would normally perform on such a system would be as follows:

- 1** Managing the physical server itself. This includes the usual server administration tasks like user and group management, file management, services management, etc.
- 2** Managing virtual servers through the interface provided by the corresponding virtualization product. The common tasks include creating virtual servers, starting and stopping a server, configuring and monitoring existing servers, and other virtualization technology-specific tasks.
- 3** Managing existing virtual servers from the inside, as if they were real physical machines (which they are in a sense). The tasks in this category are exactly the same as the tasks described in (1) above.

Generally speaking, VZAgent installed in a particular Environment can directly access that Environment only. Therefore, to perform the tasks described in (1) and (2) from the list above, you have to establish a connection with VZAgent running in the host Environment. Managing a virtual Environment from the inside, as described in (3), requires establishing a direct connection with that Environment. Imagine now that you have hundreds of Virtuozzo VE's running on the same host, so, for instance, to upload a file to each VE, you would have to establish a separate connection with every single one of them. This is where request redirection comes in. Instead of establishing a separate connection with a VE from your client program, you can simply specify the target Environment when making a call, and VZAgent will redirect the request to that Environment through its own internal mechanisms. As a result, the request will be processed on the target Environment, and the results (if any) will be sent back to the client program that initiated the original request. For this to work, you have to make sure that the necessary VZAgent components are installed on the target Environment. These components are automatically installed in every Virtuozzo VE, and can be optionally installed in the virtual Environments of other types.

Request redirection is not limited to the Host/VE scenario. You can also use it to redirect the request from one physical Environment to another physical Environment in a cluster. For more information on VZAgent cluster, please refer to the **Virtuozzo SDK Programmer's Guide**.

The following sections show how to use the request redirection feature in your client programs.

In This Chapter

1. Managing Files and Directories	79
2. Managing Operating System Services	85

1. Managing Files and Directories

This section shows how to implement the following functionality:

- Listing files and directories
- Uploading a file
- Downloading a file
- Creating a new directory

Our sample program will be able to perform all of the above tasks on the physical machine that we are connected to, and on the virtual Environments hosted by it.

Declare a new function:

```
int fileManagement(eid_t eid);
```

The function will display a simple menu to the user, allowing to select the task to perform. Each task will accept the user input (path, file name, directory name, etc.) and will display the results on the screen.

First, let's declare some variables.

```
// A container to store the filesystem path
// that the user will specify.
VZLFiler::PathList pathList;

// Used as output. Contains the file list.
VZLFileList outFiles;

// User input, integer value.
int iIn;

// User input, string value.
string strIn;

// Source file name for upload/download.
string srcFile;

// Destination file name for upload/download.
string dstFile;

// List of files to upload.
VZLFilerBase::UploadInfoList uploadFileList;

// File upload info.
VZLFileUploadFromFileSP uploadInfo;

// List of files to download.
VZLFilerBase::DownloadInfoList downloadFileList;

// File download info.
VZLFileDownloadToFileSP downloadInfo;

// Options used during file upload/download.
VZLFilerOptions filerOptions;
```

Create `VLZFiler`, the main object providing the file management interface. As with other interfaces, use object factory to create it.

```
VZLFilerSP vzlFiler = VZLLibFunctionality::kit().getFiler(  
    eid, accessPoint);
```

The `getFiler` method call above is exactly the place where the *request redirection* that we discussed in this chapter's introduction is used. The `eid` parameter specifies the ID of the Environment that you want to manage files for. To better understand the meaning of this, compare it to the VE management functionality described in the **Starting, Stopping, Restarting an Environment** section (see page 26). Notice that in that section, when we create the Environment management object (`VZAEEnvM`) using object factory, we pass the root Environment ID to the `getVZAEEnvM` method, NOT the EID of the VE that we want to stop, start, or restart. The EID of the VE that we want to stop, start, or restart is passed to the appropriate function that performs the actual operation. As a result, the request is processed on the host Environment by `VZAgent` invoking the appropriate *Virtuozzo* utility, which actually performs the specified operation on the specified *Virtuozzo* VE. Not so in the file management case. Here, we pass the ID of the target Environment to the `getFiler` method when we create the file management object, thus instructing `VZAgent` running in the root Environment to *redirect* all subsequent file management requests to `VZAgent` running in the target Environment. As a result, all calls made through this interface will be executed on the target Environment. The redirection is performed internally by `VZAgent` and does not require any additional steps.

When passing Environment ID to the `getFiler` method, use the following guidelines:

- 1 If you want to manage files on the root Environment, pass the root Environment ID.
- 2 If you want to manage files on one of the *Virtuozzo* VEs, pass the EID of that VE (not to be confused with the internal *Virtuozzo* VEID).
- 3 If you have `VZAgent` cluster, and you want to manage files on one of the slave Environments, pass the EID of that slave Environment (network traffic considerations apply when redirecting calls across cluster). Alternately, you can connect to the slave Environment directly (the Environment becomes the root Environment for the connection) and manage files on it without request redirection. For more info on `VZAgent` cluster, see *Virtuozzo SDK Programmer's Guide*.

Here's the main File Management program loop.

```
while (true)
{
```

Display the user menu.

```
    cout << endl;
    cout << "File Management menu" << endl << endl;
    cout << "1. List files" << endl;
    cout << "2. Upload a file" << endl;
    cout << "3. Download a file" << endl;
    cout << "4. Create a new directory" << endl;
    cout << endl;
```

Allow the user to select a menu item.

```
    cout << "Enter menu item number (any non-numeric key to return to
the main menu): ";
    if (!(cin >> iIn))
    {
        cin.clear();
        cin.ignore(10000, '\n');
        return -1;
    };
```

Based on the user selection, perform the appropriate task.

```
    switch (iIn)
    {
```

The "List Files" menu item.

```
        case 1:
```

Accept an input from the user (the path).

```
            cout << "Path: ";
            cin >> strIn;
            pathList.clear();
            pathList.push_back(strIn);
```

Get the list of files and directories in the specified location.

```
            outFiles.clear();
            if (vzlfiler->list(pathList, &outFiles) != 0)
            {
                GetLastError();
                system("cls");
                continue;
            };
            cout << endl;
```

Display the retrieved list on the screen. Only some of the most commonly used properties are displayed here. See Programmer's Reference for the complete list of properties.

```
            cout << "Type\t\t" << "Size\t" << "Name" << endl;
            cout << string(65, '-') << endl;
```

Iterate through the container.

```
            for (VZLFileList::iterator iter = outFiles.begin();
                iter != outFiles.end(); ++iter)
            {
```

Determine the type of the current filesystem element.

```

switch (VZLFileTypeSP(*iter)->type)
{
case FTYPE_FIFO:
    cout << "FIFO          ";
    break;
case FTYPE_CHARDEV:
    cout << "character device";
    break;
case FTYPE_DIR:
    cout << "directory      ";
    break;
case FTYPE_BLOCKDEV:
    cout << "block device   ";
    break;
case FTYPE_REGULAR:
    cout << "file           ";
    break;
case FTYPE_SYMBOLIC:
    cout << "symbolic link   ";
    break;
case FTYPE_SOCKET:
    cout << "socket          ";
    break;
case FTYPE_REMOVABLE:
    cout << "removable drive ";
    break;
case FTYPE_HARDDISK:
    cout << "hard disk drive ";
    break;
case FTYPE_REMOTE:
    cout << "remote drive    ";
    break;
case FTYPE_CDROM:
    cout << "CD-ROM          ";
    break;
case FTYPE_RAM:
    cout << "RAM             ";
    break;
case FTYPE_LOOPBACK:
    cout << "loopback file   ";
    break;
default:
    cout << "N/A            " << endl;
    break;
}

```

Get the size of the current element.

```
cout << VZLFileTypeSP(*iter)->size << "\t";
```

Get the name of the element.

```

    cout << VZLFileTypeSP(*iter)->name << endl;
}
cout << endl;
break;

```

The "Upload a file" menu item.

```
case 2:
```

Allow the user to input the source and destination file information. The path must be specified using forward slashes on both Windows and Linux.

```
cout << "Source file name (e.g. c:/myfile.txt): ";
cin >> srcFile;
cout << "Destination file name (e.g. /root/myfile.txt): ";
cin >> dstFile;
```

Create and populate the necessary objects. The `uploadFileList` object is a vector. To upload more than one file in a single call, add as many elements as needed.

```
uploadInfo =
    VZLFileUploadFromFileSP(
        new VZLFileUploadFromFile(srcFile, dstFile));
uploadFileList.clear();
uploadFileList.push_back(uploadInfo);
```

Use the packet mode transport. This is the recommended transport type for file uploading and downloading.

```
filerOptions.options = VZLFilerOptions::PACKET_MODE;
```

Finally, upload the selected file(s).

```
if (vzlFiler->upload(uploadFileList, filerOptions) != 0)
{
    getLastError();
    system("cls");
    continue;
};
break;
```

The "Download a file" menu item.

```
case 3:
```

Allow the user to input the source and destination file information. The path must be specified using forward slashes on both Windows and Linux.

```
cout << "Source file name (e.g. /root/myfile.txt): ";
cin >> srcFile;
cout << "Destination file name (e.g. c:/myfile.txt): ";
cin >> dstFile;
```

Create and populate the necessary objects. The `downloadFileList` object is a vector. To download more than one file in a single call, add as many elements as needed.

```
downloadInfo =
    VZLFileDownloadToFileSP(
        new VZLFileDownloadToFile(srcFile, dstFile));
downloadFileList.clear();
downloadFileList.push_back(downloadInfo);
```

Use the packet mode transport.

```
filerOptions.options = VZLFilerOptions::PACKET_MODE;
```

Download the file(s).

```
if (vzlFiler->download(downloadFileList, filerOptions) != 0)
{
    getLastError();
    system("cls");
    continue;
};
break;
```

The "Create a new directory" menu item.

```
case 4:
```

Allow the user to input a name for the new directory. The path must be specified using forward slashes on both Windows and Linux.

```
cout << "Directory name to create (e.g. /root/mydir): ";  
cin >> dstFile;  
pathList.push_back(dstFile);
```

Create the specified directory.

```
        vzlFiler->mkdir(pathList);  
        break;  
default:  
    system("cls");  
    continue;  
    break;  
}  
}
```

2. Managing Operating System Services

The operating system service management interface allows you to retrieve the list of services and service details, to stop and restart a service, to set runlevels and the service startup type.

Using the service management interface is similar to using the file management interface described in the previous section. First, you use object factory to create the `VZLServiceM` object, which provides methods for service management. The ID of the target Environment must be passed to the factory's `getServiceM` method. You then simply call the appropriate method to perform a desired task passing the name of a services to it. On Linux systems, when starting, stopping, or restarting a service, you also have to specify whether the selected service depends on `xinted`.

The following code sample shows how to perform the following tasks:

- Get a list of services from the selected Environment.
- Get the specified service details.
- Start a service.
- Stop a service.

```
int servicesManagement(eid_t eid)
{
    int iIn;
    string sIn;

    // Create the VLZServiceM object (service management) using
    // object factory.
    // The "eid" parameter specifies the target Environment.
    VZLServiceMSP vzlService = VZLLibFunctionality::kit().getServiceM(
        eid, accessPoint);

    // Services Management main loop.
    system("cls");
    while (true)
    {
        // Display the user menu.
        cout << endl;
        cout << "Servies Management menu" << endl << endl;
        cout << "1. List services" << endl;
        cout << "2. Service details" << endl;
        cout << "3. Start a service" << endl;
        cout << "4. Stop a service" << endl;
        cout << endl;

        // Allow the user to select a menu item.
        cout << "Enter menu item number (any non-numeric key to return
to the main menu): ";
        if (!(cin >> iIn))
        {
            cin.clear();
            cin.ignore(10000, '\n');
            return -1;
        }
    }
}
```

```

};

// Used as an oputput
// Contains the list of services.
VZLSERVICELIST serviceList;

// Service details.
VZLSERVICE serviceDetails;

switch (iIn)
{
// List services.
case 1:
    cout << "Please wait...";
    if (vzlService->getList(&serviceList) != 0)
    {
        getLastError();
        system("cls");
        continue;
    }

    // Display the retrieved services on the screen.
    cout << endl << endl;
    cout << setiosflags(ios::left);
    cout << setw(20);
    cout << "Name" << "State" << endl << endl;

    for (VZLSERVICELIST::iterator iter = serviceList.begin();
        iter != serviceList.end(); ++iter)
    {
        cout << setiosflags(ios::left);
        cout << setw(20);

        // Service name.
        cout << iter->name;

        // Service state.
        switch (iter->state)
        {
        case 0:
            cout << "stopped";
            break;
        case 1:
            cout << "started";
            break;
        case -1:
            cout << "N/A";
            break;
        }
        cout << endl;
    }
    break;

// Service details.
case 2:
    cout << "Enter service name: ";
    if (!(cin >> sIn))
    {
        cin.clear();
        cin.ignore(10000, '\n');
    }
}

```

```
        return -1;
    };

    // Get service details.
    if (vzlService->get(sIn.c_str(), &serviceDetails) != 0)
    {
        getLastError();
        system("cls");
        continue;
    };
    cout << "Service:      " << serviceDetails.name << endl;
    cout << "Description: " <<
        (serviceDetails.description == "" ? "N/A" :
         serviceDetails.description);
    cout << endl;
    break;

// Start a service.
case 3:
    serviceDetails = VZLService();
    cout << "Enter service name: ";
    if (!(cin >> serviceDetails.name))
    {
        cin.clear();
        cin.ignore(10000, '\n');
        return -1;
    };

    // On Linux, must specify if this is a xinetd service.
    cout << "Is this a xinetd service (Y/N)? ";
    if (!(cin >> sIn))
    {
        cin.clear();
        cin.ignore(10000, '\n');
        return -1;
    };

    if (sIn == "Y" || sIn == "y")
    {
        serviceDetails.xinetd = true;
    }

    // Start a service.
    if (vzlService->start(serviceDetails) != 0)
    {
        getLastError();
        system("cls");
        continue;
    };
    break;

// Stop a service.
case 4:
    serviceDetails = VZLService();
    cout << "Enter service name: ";
    if (!(cin >> serviceDetails.name))
    {
        cin.clear();
        cin.ignore(10000, '\n');
        return -1;
    };
```

```
};

// On Linux, must specify if this is a xinetd service.
cout << "Is this a xinetd service (Y/N)? ";
if (!(cin >> sIn))
{
    cin.clear();
    cin.ignore(10000, '\n');
    return -1;
};

if (sIn == "Y" || sIn == "y")
{
    serviceDetails.xinetd = true;
}
if (vzlService->stop(serviceDetails) != 0)
{
    getLastError();
    system("cls");
    continue;
};
break;
default:
    system("cls");
    continue;
    break;
}
}
```


CHAPTER 4

Appendix A Sample Program

```

// VzSDK-C++Tutorial : Defines the entry point for the console
// application.
//

#include "stdafx.h"
#include <tchar.h>
#include <iostream>
#include <iomanip>
#include <VZLLibAgent/VZLAccessPoint.h>
#include <VZLFunctionality/VZLLibFunctionality.h>
#include <VZLFunctionalityAgent/VZLFunctionalityAgentMain.h>
#include <VZLFunctionality/VZLFunctionalityMain.h>
#include <VZLFunctionality/VZLEnvSampleM.h>
#include <VZAFFunctionality/VZAFFunctionalityMain.h>
#include <VZAFFunctionalityAgent/VZAFFunctionalityAgentMain.h>
#include <VZAFFunctionality/VZAEVEnvM.h>
#include <VZAFFunctionality/VZAPkgM.h>
#include <VZAFFunctionality/VZALibFunctionality.h>
#include <VZLFunctionality/VZLAlertM.h>

#include <VZLFunctionality/VZLPerfMon.h>
#include <VZLFunctionalityAgent/VZLMonitorAgent.h>

using namespace std;
using namespace VZL;
using namespace VZA;

// Access Point object.
VZLAccessPointSP accessPoint;

// EID list.
typedef pair<eid_t, string> EIDNamePair;
typedef vector<EIDNamePair> EIDNameList;

string getEnvStatusDesc(const VZLEnvStatus& envStatus);
int displayEnvTree(VZLAccessPointPrototype::EnvValuesListSP envList,
eid_t parentEID, EIDNameList & eids, int indent);
EIDNamePair selectEnvFromTree();
int createVirtuozzoVE();
int startVirtuozzoVE(eid_t eid);
int stopVirtuozzoVE(eid_t eid);
int restartVirtuozzoVE(eid_t eid);
int destroyVirtuozzoVE(eid_t eid);

int displayEnvInfo(eid_t eid);
int displayGenericEnvInfo(eid_t eid);
int displayVirtuozzoVEinfo(eid_t eid);
int modifyVirtualConfig(eid_t eid);
int modifyVirtuozzoVEConfig(VZLEnvCSP env);

// Environment monitoring.
int selectPerformanceCounters(string &classOut, string &counterOut);

```

```

// Performance Monitoring
class PerfMonHandler : public VZLMonitorDataHandlerPrototype
{
public:
    void PerfMonHandler::handleOk()
    {
        // Invoked on Monitor start and finish.
    }

    void handleError(const VZLRequestErrorData &err)
    {
        cout << endl;
        cout << err.code << ", " << err.message.c_str();
        cout << endl;
    }

    void processData(const VZLMonitorDataList& datalist)
    {
        // Get the data from the container
        // and display it on the screen.
        for(VZLMonitorDataList::const_iterator it =
            datalist.begin(); it != datalist.end(); ++it)
        {
            const VZLMonitorData& data = *it;

            // First, display the environment ID,
            // the class name, the beginning and the end
            // time of the report.
            cout << endl;
            cout << "EID:\t" << data.eid.toString() << endl;
            cout << "Class:\t" << data.counterClass << endl;
            cout << "From:\t" << ctime(&data.interval.startTime);
            cout << "To:\t" << ctime(&data.interval.endTime) << endl;

            cout << "\t\tmin\tmax\tavg\tcur" << endl << endl;

            // Now, display the actual counter data.
            for(VZLMonitorData::InstanceMap::const_iterator
                inst = data.counters.begin();
                inst != data.counters.end(); ++inst)
            {
                cout << "Instance:\t" << inst->first << endl;
                for(VZLMonitorData::CountersMap::const_iterator
                    counter =
                        inst->second.begin();
                    counter != inst->second.end();
                    ++counter)
                {
                    cout << "\t\t" << counter->first << ":" << endl;
                    cout << "\t\t" << counter->second.min.intval <<
                        "\t";

                    cout << counter->second.max.intval << "\t";
                    cout << counter->second.avg.intval << "\t";
                    cout << counter->second.cur.intval << endl
                        << endl;
                }
            }
            cout << endl;
        }
        cout << string(65, '=') << endl;
    }
}

```

```

    }

protected:
    void defaultHandleError(int code)
    {
        cout << endl;
        cout << "Error " << code << endl;
        cout << endl;
    };
};

typedef intrusive_smart_pointer<PerfMonHandler> PerfMonHandlerSP;
PerfMonHandlerSP perfMonHandler(new PerfMonHandler());

int getPerformanceReport(eid_t eid);
int startPerfMon(PerfMonHandlerSP perfMonHandler, eid_t eid);

//////////
// Events

// Status Event Handler.
class StatusEventHandler: public VZLEnvStatusEventReceiver
{
public:
    // StatusEventHandler(){};
    // ~StatusEventHandler(){};

    // Called from the background thread when
    // an event notification is received.
    // The "event" object contains the base event information,
    // such as subscription name, event ID, EID of the
    // environment, as well as the previous environment
    // state and transition code values, and
    // and the new state and transition codes.
    void handleEvent(const VZLEnvStatusEvent &event)
    {
        cout << endl;
        cout << string(65, '-') << endl;
        cout << "VE status change detected:" << endl;
        cout << "Subscription name: " << event.subscriptionName <<
endl;
        cout << "Event category: " << event.category << endl;
        VZLInfo info;
        event.toInfo(info);
        cout << "Message: " << info.toString() << endl;
        cout << "Old status: " << getEnvStatusDesc(event.oldStatus) <<
endl;
        cout << "New status: " <<
getEnvStatusDesc(event.newStatus.state) << endl;
        cout << string(65, '-') << endl;
    };
};

VZLEnvStatusEventReceiverSP eventHandler(new StatusEventHandler());
VZLEnvStatusEventSubscriberSP statusSubscriber;

// Configuration change event handler.
class ConfigEventHandler: public VZLEnvConfigEventReceiver
{
public:
    // Called from the background thread when
    // an event notification is received.

```

```

// The "event" parameter contains the event information.
void handleEvent(const VZLEnvConfigEvent &event)
{
    cout << endl;
    cout << string(65, '-') << endl;
    cout << "VE config change detected:" << endl;
    cout << "Subscription name: " << event.subscriptionName <<
endl;
    cout << "Event category: " << event.category << endl;
    VZLInfo info;
    event.toInfo(info);
    cout << "Message: " << info.toString() << endl;

    // The event.config member contains the regular
    // configuration information. The event.vconfig
    // member contains the virtual configuration information.
    // Depending on your needs, you may examine the values
    // contained in these objects and take some
    // action if necessary.
    // Please see Getting an Environment Configuration for
    // the examples.
    cout << string(65, '-') << endl;
};

VZLEnvConfigEventReceiverSP configHandler(new ConfigEventHandler());
VZLEnvConfigEventSubscriberSP configSubscriber;

////////////////////////////////////////
// Alerts

int getAlertList();

class AlertHandler: public VZLAlertDataReceiver
{
public:

    // This method will be automatically called by the
    // background thread as soon as the alert is raised on
    // an Environment, and the alert data is received from
    // the server.
    void handleEvent(const VZLAlertData &data)
    {
        VZLInfo info;
        data.toInfo(info);
        cout << endl;
        cout << string(65, '-') << endl;

        // Display the alert type on the screen.
        switch (data.type)
        {
            case VZLAlertData::AT_GREEN:
                cout << "GREEN ALERT";
                break;
            case VZLAlertData::AT_YELLOW:
                cout << "YELLOW";
                break;
            case VZLAlertData::AT_RED:
                cout << "RED ALERT";
                break;
        }
    }
};

```

```

        case VZLAlertData::AT_BLACK:
            cout << "BLACK";
            break;
        default:
            cout << "ALERT (unknown type)";
            break;
    }

    // Display the rest of the alert information.
    cout << endl << endl;

    // The ID of the affected Environment.
    cout << "EID: " << data.eid.toString() << endl;

    // Subscription name.
    cout << "Subscription: " << data.subscriptionName << endl;

    // A text message describing the alert.
    cout << "Message: " << info.toString() << endl;
    cout << string(65, '-') << endl;
    cout << endl;
    };
};

// Create the alert handler object.
VZLAlertDataReceiverSP alertHandler(new AlertHandler());
VZLAlertDataSubscriberSP alertSubscriber;

////////////////////////////////////
// Main Program

int _tmain(int argc, _TCHAR* argv[])
{
    // VZAgent server IP address.
    string address = "192.168.0.151";

    // Login.
    string login = "vzadmin";

    // Password.
    string password = "1q2w3e";

    // Create the Connection Info object.
    VZLConnectionInfo connInfo(address, login, password);

    // Create the Access Point object.
    accessPoint = VZLAccessPointPrototype::createInstance();

    // Establish a connection with VZAgent server.
    if (accessPoint->initializeSync(connInfo,
        NULL, CACHE_ALL | CACHE_TREE) != 0)
    {
        cout << "Error, unable to connect to VZAgent." << endl;
        cout << "Please check the connection parameters." << endl;
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error ( " << errorData.code << " ) ";
        cout << errorData.message << endl;
        return -1;
    }
}

```

```

// Initialize the generic functionality plugin.
initVZLFunctionalityAgent();

// Initialize the VZA (Virtuozzo) functionality plugin.
initVZAFunctionalityAgent();

EIDNamePair eidNamePair;
eidNamePair.first = accessPoint->getRootEID();
eidNamePair.second = "root";

// Main program loop
int in;
while (true)
{
    cout << endl;
    cout << string (65, '=') << endl << endl;
    cout << "MAIN MENU" << endl << endl;
    cout << string (65, '=') << endl;
    cout << "Selected Environment: " << eidNamePair.second <<
endl;
    cout << endl;
    cout << " 0. Display this menu" << endl;
    cout << " 1. Display Environment list" << endl;
    cout << " 2. Create new Environment" << endl;
    cout << " 3. Start" << endl;
    cout << " 4. Stop" << endl;
    cout << " 5. Restart" << endl;
    cout << " 6. Get Environment info" << endl;
    cout << " 7. Modify virtual configuration" << endl;
    cout << " 8. Destroy Environment" << endl;
    cout << " 9. Get Performance Report" << endl;
    cout << "10. Start Performance Monitor" << endl;
    cout << "11. Stop Performance Monitor" << endl;
    cout << "12. Subscribe to status change events" << endl;
    cout << "13. Unsubscribe from status change events" << endl;
    cout << "14. Subscribe to config change events" << endl;
    cout << "15. Unsubscribe from config change events" << endl;
    cout << "16. Get Alerts" << endl;
    cout << "17. Subscribe to alerts" << endl;
    cout << "18. Unsubscribe from alerts" << endl;
    cout << endl;
    cout << "Enter menu item number (any alpha key to exit): ";

    while (true)
    {
        if (!(cin >> in))
        {
            cout << "Invalid entry. Please try again" << endl;
            cin.clear();
            cin.ignore(10000, '\n');
            return 0;
        };
        switch (in)
        {
            case 0:
                break;
            case 1:
                system("cls");
                eidNamePair = selectEnvFromTree();

```

```

        system("cls");
        break;
    case 2:
        createVirtuozzoVE();
        break;
    case 3:
        startVirtuozzoVE(eidNamePair.first);
        break;
    case 4:
        stopVirtuozzoVE(eidNamePair.first);
        break;
    case 5:
        restartVirtuozzoVE(eidNamePair.first);
        break;
    case 6:
        system("cls");
        displayEnvInfo(eidNamePair.first);
        system("pause");
        break;
    case 7:
        modifyVirtualConfig(eidNamePair.first);
        break;
    case 8:
        destroyVirtuozzoVE(eidNamePair.first);
        break;
    case 9:
        getPerformanceReport(eidNamePair.first);
        system("pause");
        break;
    case 10:
        startPerfMon(perfMonHandler, eidNamePair.first);
        break;
    case 11:
        if (perfMonHandler->stop() == 0)
        {
            cout << "Monitor has been stopped." << endl;
        }
        break;
    case 12:
        // Obtain a pointer to the subscriber object.
        if (!statusSubscriber)
        {
            statusSubscriber = accessPoint-
>getEnvStatusEventSubscriber();
        }
        // Subscribe to status changes events.
        if (statusSubscriber->subscribe(eventHandler, -200) !=
0)
        {
            cout << "Error, unable to subscribe." << endl;
        }
        break;
    case 13:
        statusSubscriber->unsubscribe(eventHandler);
        break;
    case 14:
        // Obtain a pointer to the subscriber object.
        if (!configSubscriber)
        {

```

```

        configSubscriber = accessPoint-
>getEnvConfigEventSubscriber();
    }
    // Subscribe to configuration changes events.
    if (configSubscriber->subscribe(configHandler, -200)
!= 0 )
    {
        cout << "Error, unable to subscribe." << endl;
    }
    break;
case 15:
    configSubscriber->unsubscribe(configHandler);
    break;
case 16:
    getAlertList();
    system("pause");
    break;
case 17:
    // Obtain a pointer to the subscriber object.
    if (!alertSubscriber)
    {
        alertSubscriber = accessPoint-
>getAlertSubscriber();
    }
    // Subscribe to alerts.
    if (alertSubscriber->subscribe(alertHandler, -200) !=
0)
    {
        cout << "Error, unable to subscribe to alerts." <<
endl;
    }
    break;
case 18:
    alertSubscriber->unsubscribe(alertHandler);
    break;
default:
    cout << "Invalid entry. Please try again" << endl;
    continue;
}
break;
}
}
return 0;
}

/*****

// Here we have two functions that build the Environment tree.
// One function will actually build and display the
// tree (it will be called recursively).
// The second function will initially call the first
// function and will allow the user to select
// an Environment from the tree.

// Builds and displays the Environment tree.
// Parameters
// envList : The Environment list retrieved from
// the Access Point cache.
// parentEID : Parent Environment ID (the top level
// of this branch).

```



```

// eids :      The list of the retrieved EID/Name pairs.
// indent :    An indentation at which the Environment name
//             should be displayed.

int displayEnvTree(VZLAccessPointPrototype::EnvValuesListSP
                  envList, eid_t parentEID,
                  EIDNameList& eids, int indent)
{
    // The Environment container iterator.
    // The iterator's member access operator ->() returns
    // a pointer to the VZLEnv object, containing
    // an individual Environment information.
    VZLEnvCache::EnvValuesList::const_iterator cit;

    // Regular configuration structure.
    VZLEnvConfig envConfig;

    // Virtuozzo virtual config structure.
    VZAEnvConfig vzEnvConfig;

    // List of the Environment IP addresses.
    VZLEnvConfig::IPAddressList IPList;

    // Environment type.
    VZLEnvConfigBasic::type_t envType;

    // Environment name.
    string envName;
    // Hostname
    string envHostname;
    // IP address.
    string IPaddress;
    // Environment title.
    string envTitle;
    // Environment index (for display)
    int envIndex;

    // Iterate through the Environment container.
    for (cit = envList->begin(); cit != envList->end(); ++cit)
    {
        // The function actually builds one tree branch
        // at a time. Each Environment in the list has
        // the parent EID property, which indicates the
        // parent Environment EID. You can use this property
        // to determine whether an Environment belongs to
        // the current branch. If does, add it to the branch.
        if (cit->getParentEID() == parentEID)
        {
            // some tree formatting...
            cout << string(indent, ' ') << "| " << endl;
            cout << string(indent, ' ') << "|-- ";

            // Based on the Environment type, use the
            // correct configuration structure and then
            // read the Environment configuration info
            // from it.
            envType = cit->getType();

            // Generic environment.
            if (envType == "generic")

```

```

{
    envConfig = cit->getConfig();
    envName = envConfig.getName().c_str();
    envHostname = envConfig.getHostname().c_str();
    envConfig.getAddresses(IPList);
    if (IPList.isSpecified() && IPList->size() != 0)
    {
        IPaddress = IPList->begin()->ip;
    }
}
// Virtuozzo Virtual Environment.
else if (envType == "virtuozzo")
{
    vzEnvConfig = cit->getVirtualConfig();
    envName = vzEnvConfig.getName().c_str();
    envHostname = vzEnvConfig.getHostname().c_str();
    vzEnvConfig.getAddresses(IPList);
    if (IPList.isSpecified() && IPList->size() != 0)
    {
        IPaddress = IPList->begin()->ip;
    }
}
else
{
    // This is a placeholder for future types.
    // If you have other Environment types,
    // use the correct config structure here.
}

// There are several fields that can be used
// as the Environment "name". None of those
// fields are mandatory, however. Let's check
// each field and use one that has a value as
// the Environment "Title".
envTitle = envName;
if (envTitle.empty())
{
    // Hostname
    envTitle = envHostname;
    if (envTitle.empty())
    {
        // IP address
        envTitle = IPaddress;
        if (envTitle.empty())
            envTitle = "N/A";
    }
}

// Insert the current EID/Name pair into
// the list. The list will be used later
// to determine the Environment title based on
// the ID (for display purposes).
EIDNamePair pair;
pair.first = cit->getEID();
pair.second = envTitle;
eids.push_back(pair);

// Display the basic Environment properties
// on the screen.
// Title:

```

```

        envIndex = int(eids.size()) - 1;
        cout << "(" << envIndex << ") " << envTitle;

        // some screen output formatting...
        char buffer[30];
        itoa(envIndex, buffer, 10);
        cout << string((33 - (strlen(buffer) +
envTitle.length()))), ' ');

        // Environment type:
        cout << envType.c_str() << "\t";

        // Environment status.
        cout << getEnvStatusDesc(cit->getStatus()) << endl;

        // Go to the next level of recursion to
        // build the next tree level.
        indent = indent + 3;
        displayEnvTree(envList, cit->getEID(), eids, indent);
        indent = indent - 3;
    }
}
return 0;
}

// The second function (works together with
// displayEnvTree() listed above)
// Allows to select an Enviroment from the tree.
EIDNamePair selectEnvFromTree()
{
    // Get the Environment list from Access Point cache.
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();

    // Display heading.
    cout << "Environments\t\t\t\tType\t\tStatus" << endl;
    cout << string(65, '-') << endl;

    // Get the root Environment information.
    VZLEnvCSP env = envList->getEnv(accessPoint->getRootEID());

    // A list of EIDs.
    EIDNameList eids;
    // The EID/Name pair.
    EIDNamePair eidNamePair;
    // Popoulate the pair with root Environment values.
    eidNamePair.first = accessPoint->getRootEID();
    eidNamePair.second = "root";
    eids.push_back(eidNamePair);
    cout << "(0) root - " <<
        env->getConfig().getHostname().c_str() << endl;

    // Build and display the Environment tree,
    // starting with the root Environment.
    // The function will be called recursively for each
    // Environment in the list, because each Environment
    // may have child Environments of its own.
    displayEnvTree(envList, accessPoint->getRootEID(), eids, 0);

    // Allow the user to select an environment from

```

```

// the list. The user must select the Environment
// index number (a sequential number added to
// each Environment entry by the displayEnvTree() function).
// The Environment EID/Name pair is the element of the
// eids array with the selected index.
unsigned int iIn;
while (true)
{
    cout << endl;
    cout << "Enter Environment index number: ";
    cin >> iIn;

    if (iIn >= eids.size() || iIn < 0)
    {
        cout << "Invalid entry, please try again." << endl;
        iIn = 0;
        continue;
    }

    return eids[iIn];
}
}

/*****
// A helper function to convert the Environment state and
// transition codes into a description that can be displayed
// to the user. The Environment state and transition are
// enumerations defined in the VZAgent client library.
string getEnvStatusDesc(const VZLEnvStatus& envStatus)
{
    string statusDesc;

    // Determining the VE state and transition.
    if (envStatus.transition == ENV_TRANS_NONE)
    {
        // VE is not in transition. Display status.
        switch (envStatus.state)
        {
            case ENV_STAT_UNEXIST:
                statusDesc = "dropped";
                break;
            case ENV_STAT_DOWN:
                statusDesc = "stopped";
                break;
            case ENV_STAT_MOUNTED:
                statusDesc = "mounted";
                break;
            case ENV_STAT_SUSPENDED:
                statusDesc = "suspended";
                break;
            case ENV_STAT_RUNNING:
                statusDesc = "running";
                break;
            default:
                statusDesc = envStatus.state;
                break;
        }
    }
    else
    {

```

```

        // VE is in transition (its stable state is changing).
        switch (envStatus.transition)
        {
            case ENV_TRANS_CREATING:
                statusDesc = "creating ...";
                break;
            case ENV_TRANS_MOUNTING:
                statusDesc = "mounting ...";
                break;
            case ENV_TRANS_STARTING:
                statusDesc = "starting ...";
                break;
            case ENV_TRANS_STOPPING:
                statusDesc = "stopping ...";
                break;
            case ENV_TRANS_DESTROYING:
                statusDesc = "destroying ...";
                break;
            case ENV_TRANS_BACKING_UP:
                statusDesc = "backing up ...";
                break;
            case ENV_TRANS_RESTOREING:
                statusDesc = "restoring ...";
                break;
            case ENV_TRANS_SUSPENDING:
                statusDesc = "suspending ...";
                break;
            case ENV_TRANS_RESUMING:
                statusDesc = "resuming ...";
                break;
            default:
                statusDesc = envStatus.transition;
        }
    }

    return statusDesc;
}

/*****

int createVirtuozzoVE()
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAFunctionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "ERROR, unable to create VZA::VZAEnvMSP object." <<
endl;
        return -1;
    }

    // Environment ID.

```

```

eid_t envIDout;

// Environment configuration info.
VZAEEnvConfig envConf;

int iIn;
string sIn;

// VEID
cout << "VEID: ";
cin >> iIn;
envConf.setVeid(iIn);

// Environment NAME
cout << "Environment Name: ";
cin >> sIn;
envConf.setName(sIn);

// IP address
cout << "IP address: ";
cin >> sIn;
VZLEnvConfig::IPAddressList::value_type IPList;
VZLIPAddress IPaddress;
IPaddress.ip = sIn;
IPList.insert(IPaddress);
envConf.setAddresses(IPList);

// Hostname
cout << "Hostname: ";
cin >> sIn;
envConf.setHostname(sIn);

// Environment description.
cout << "Description: ";
cin >> sIn;
envConf.setDescription(sIn);

// OS template.
// Frist, get the list of the installed
// standard OS templates.

cout << endl << "Available OS templates:" << endl;

VZA::VZATEMSP vzatem = VZALibFunctionality::kit().getVZATEM(
    accessPoint->getRootEID(),
    accessPoint,
    VZA::vzaFunctionalType);

if (!vzatem)
{
    cerr << "ERROR, unable to create VZATEMSP object." << endl;
    return -1;
}

VZAPackageInfoList pkgInfoList;
eid_t eid;

// Passing empty eid, which means that
// we want the templates installed in the
// root Environment.

```

```

if (vzatem->ls(eid, &pkgInfoList, "os") != 0)
{
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error ( " << errorData.code << " ) ";
    cout << errorData.message << endl << endl;
}

bool bPkgListEmpty = true;
std::vector<string> pkgList;
int ctr = 0;

if (pkgInfoList.size() != 0)
{
    for (VZAPackageInfoList::iterator pit = pkgInfoList.begin();
        pit != pkgInfoList.end(); ++pit)
    {
        pkgList.push_back(pit->name);
        cout << "(" << ctr++ << " ) ";
        cout << pit->name << "\tstandard template" << endl;
    }
}

// Now, get the list of the installed
// EZ templates.
VZA::VZAPkgMSP vzapkgm = dynamic_pointer_cast<VZAPkgM>(
    VZLLibFunctionality::kit().getPkgM(
        accessPoint->getRootEID(),
        accessPoint,
        VZA::vzaFunctionalType));

if (!vzapkgm)
{
    cerr << "ERROR, unable to create VZAPkgMSP object." << endl;
    return -1;
}

VZLPackageList pkgOut;
vector<string> pkgType;
pkgType.push_back("os");
VZLPackageManagerListOptions pkgOptions;
pkgOptions.types = pkgType;

vzapkgm->list(accessPoint->getRootEID(),
    &pkgOut, NULL, pkgOptions);

if (pkgOut.size() == 0 && ctr == 0)
{
    cout << "No OS templates were found." << endl;
    system("pause");
    return -1;
}

for (VZLPackageList::iterator pit = pkgOut.begin();
    pit != pkgOut.end(); ++pit)
{
    pkgList.push_back((*pit)->name);
    cout << "(" << ctr++ << " ) ";
    cout << (*pit)->name << "\tEZ Template" << endl;
}

```

```

cout << endl;

cout << "Enter template number: ";

cin >> iIn;

envConf.setOsTemplate(VZATemplate(pkgList[iIn]));

// Sample configuration.
initVZLFunctionalityAgent();
VZLEnvSampleMSP vzlSampleConf =
    VZLLibFunctionality::kit().getEnvSampleM(
        accessPoint->getRootEID(),
        accessPoint);

if (!vzlSampleConf)
{
    cerr << "ERROR, unable to create VZLEnvSampleMBaseSP object."
<< endl;
    return -1;
}

VZLSampleID sampleConfigID;
VZLSampleConfList confs;
VZLSampleIDList ids;
vzlSampleConf->getSampleConf(&confs, ids);

pair<VZLGUID, string> IdNamePair;
vector<pair<VZLGUID, string> > sampleList;
cout << endl << "Sample configuration list" << endl << endl;

ctr = 0;
for (VZLSampleConfList::iterator sit = confs.begin();
    sit != confs.end(); ++sit)
{
    IdNamePair.first = sit->id;
    IdNamePair.second = sit->name;
    sampleList.push_back(IdNamePair);
    cout << "(" << ctr++ << " ) ";
    cout << sit->name << endl;
}

cout << "Enter sample number: ";
cin >> iIn;
envConf.setBaseSampleId(sampleList[iIn].first);

// Create VE.
cout << endl << "Creating VE ...";
if (vzaenvm->create(&envIDout, envConf) != 0)
{
    VZLRequestErrorData errorData = getLastErrorData();
    cout << endl;
    cout << "Error ( " << errorData.code << " ) ";
    cout << errorData.message << endl;
    return -1;
};
cout << endl << endl << "Virtual Environment was created
successfully." << endl;
system("pause");
return 0;

```



```

}
/*****
// Displays the specified Environment information.
int displayEnvInfo(eid_t eid)
{
    // Get the Environment list from cache.
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();

    // Get the specified Environment information.
    VZLEnvCSP env = envList->getEnv(eid);

    // Get the VZLEnvConfig object containing the specified
    // Environment configuration info.
    VZLEnvConfig envConfig = env->getConfig();

    // Call the appropriate function to extract and display the
    // configuration info on the screen.
    if (env->getType() == "generic")
    {
        // Displays the generic Environment info.
        displayGenericEnvInfo(eid);
    }
    else if (env->getType() == "virtuozzo")
    {
        // Displays Virtuozzo VE info.
        displayVirtuozzoVEinfo(eid);
    }
    else
    {
        // This is a placeholder for future types.
        cout << "Unsupported Environment type." << endl;
        return -1;
    }

    return 0;
}
*****/

int displayGenericEnvInfo(eid_t eid)
{
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();
    VZLEnvCSP env = envList->getEnv(eid);
    VZLEnvConfig envConfig = env->getConfig();

    cout << endl << "Environment Information" << endl << endl;
    cout << "EID:\t\t" << eid.toString() << endl << endl;
    cout << "Name: \t\t" << envConfig.getName().c_str() << endl;
    cout << "Hostname:\t" << envConfig.getHostname().c_str() << endl;
    cout << "Domain:\t" << envConfig.getDomain().c_str() << endl;
    cout << "Type:\t\t" << env->getType().c_str() << endl;
    cout << "Status:\t\t" << getEnvStatusDesc(env->getStatus()) <<
endl;
    cout << "Architecture:\t" << envConfig.getArchitecture().c_str()
<< endl;
    cout << "Description:\t" << envConfig.getDescription().c_str() <<
endl;

    cout << endl << "Operating System:" << endl;

```

```

        cout << "    Name: " << envConfig.getOS().name.c_str() << endl;
        cout << "    Platform: " << envConfig.getOS().platform.c_str() <<
endl;
        cout << "    Version: " << envConfig.getOS().version.c_str() <<
endl;
        cout << "    Kernel: " << envConfig.getOS().kernel.c_str() << endl;

        VZLEnvConfig::IPAddressList IPList;
        envConfig.getAddresses(IPList);

        cout << endl << "IP Address:\t";
        for (VZLEnvConfig::IPAddressList::value_type::iterator cit =
            IPList->begin(); cit != IPList->end(); ++cit)
        {
            cout << cit->ip << " " << cit->netmask << " ";
        }
        cout << endl;
        return 0;
};

/*****

int displayVirtuozzoVEinfo(eid_t eid)
{
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();
    VZLEnvCSP env = envList->getEnv(eid);
    VZAEEnvConfig envConfig = env->getVirtualConfig();

    cout << endl << "Environment Information" << endl << endl;
    cout << "EID:\t\t" << eid.toString() << endl << endl;
    cout << "Name: \t\t" << envConfig.getName().c_str() << endl;
    cout << "Hostname:\t" << envConfig.getHostname().c_str() << endl;
    cout << "Domain:\t" << envConfig.getDomain().c_str() << endl;
    cout << "Type:\t\t" << env->getType().c_str() << endl;
    cout << "Status:\t\t" << getEnvStatusDesc(env->getStatus()) <<
endl;
    cout << "Architecture:\t" << envConfig.getArchitecture().c_str()
<< endl;
    cout << "Description:\t" << envConfig.getDescription().c_str() <<
endl;

    cout << endl << "Operating System:" << endl;
    cout << "    Name: " << envConfig.getOS().name.c_str() << endl;
    cout << "    Platform: " << envConfig.getOS().platform.c_str() <<
endl;
    cout << "    Version: " << envConfig.getOS().version.c_str() <<
endl;
    cout << "    Kernel: " << envConfig.getOS().kernel.c_str() << endl;

    VZLEnvConfig::IPAddressList IPList;
    envConfig.getAddresses(IPList);

    cout << endl << "IP Address:\t";
    for (VZLEnvConfig::IPAddressList::value_type::iterator cit =
        IPList->begin(); cit != IPList->end(); ++cit)
    {
        cout << cit->ip << " " << cit->netmask << " ";
    }
}

```

```

    cout << endl;

    // Virtuozzo-specific settings
    veid_t veid;
    envConfig.getVeid(veid);
    cout << "VEID:          " << veid;
    if (veid == 1)
    {
        cout << " (Service VE)";
    }
    cout << endl;

    VZLBoolOptionalProperty bOn;
    envConfig.getOnBoot(bOn);
    cout << "On-boot:          " << (bOn ? "on" : "off") << endl;

    envConfig.getOfflineManagement(bOn);
    cout << "Offline mgmt:     " << (bOn ? "on" : "off") << endl;

    string rootDir;
    envConfig.getVERoot(rootDir);
    cout << "Root directory: " << rootDir << endl;

    VZAEEnvConfig::NetVEthListAttribute netVEthList;
    envConfig.getNetVEths(netVEthList);
    VZAEEnvConfig::NetVEthListAttribute::value_type::iterator iter =
netVEthList->begin();

    cout << "Netowrk devices:" << endl;
    for (iter; iter != netVEthList->end(); ++iter)
    {
        cout << "      " << iter->toString() << endl;
    }

    cout << endl;
    return 0;
}

/*****/

int startVirtuozzoVE(eid_t eid)
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAFunctionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEEnvMSP vzaenvm = dynamic_pointer_cast<VZAEEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "ERROR, unable to create VZA::VZAEEnvMSP object." <<
endl;
        return -1;
    }
}

```

```

    cout << "Starting VE..." << endl;
    if (vzaenvm->start(eid) != 0)
    {
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error ( " << errorData.code << " ) ";
        cout << errorData.message << endl;
        return -1;
    }

    cout << endl << "VE started successfully." << endl;
    return 0;
};

/*****/

int stopVirtuozzoVE(eid_t eid)
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAF functionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "ERROR, unable to create VZA::VZAEnvMSP object." <<
endl;
        return -1;
    }

    cout << "Stopping VE..." << endl;
    if (vzaenvm->stop(eid) != 0)
    {
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error ( " << errorData.code << " ) ";
        cout << errorData.message << endl;
        return -1;
    }

    cout << endl << "VE has been stopped." << endl;
    return 0;
};

/*****/

int restartVirtuozzoVE(eid_t eid)
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAF functionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),

```

```

        accessPoint,
        VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "ERROR, unable to create VZA::VZAEnvMSP object." <<
endl;
        return -1;
    }

    cout << "Restarting VE..." << endl;
    if (vzaenvm->restart(eid) != 0)
    {
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error (" << errorData.code << ") ";
        cout << errorData.message << endl;
        return -1;
    }

    cout << "VE restarted successfully." << endl;
    return 0;
};

/*****

int destroyVirtuozzoVE(eid_t eid)
{
    // Initialize the VZA (Virtuozzo) functionality plugin.
    initVZAF functionalityAgent();

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "ERROR, unable to create VZA::VZAEnvMSP object." <<
endl;
        return -1;
    }
    if (vzaenvm->destroy(eid) != 0)
    {
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error (" << errorData.code << ") ";
        cout << errorData.message << endl;
        return -1;
    }

    cout << endl << "VE has been destroyed." << endl;
    return 0;
}

/*****/
// Modifies Virtual Environment configuration.

```

```

int modifyVirtualConfig(eid_t eid)
{
    // Get the Environment info from the Access Point cache.
    VZLAccessPointPrototype::EnvValuesListSP envList;
    envList = accessPoint->getEnvList();
    VZLEnvCSP env = envList->getEnv(eid);

    // Virtuozzo VE.
    if (env->getType() == "virtuozzo")
    {
        modifyVirtuozzoVEConfig(env);
    }
    // Normally, there's no virtual configuration for
    // generic Environments.
    else if (env->getType() == "generic")
    {
        cout << endl;
        cout << "Error: this Environment is not a virtual
Environment.";
        cout << endl;
        system("pause");
        return -1;
    }
    // Some other, currently unsupported virtualization technology.
    else
    {
        // This is a placeholder for future types.
        cout << "Unsupported virtual Environment type." << endl;
    }

    return 0;
}

/*****
// Modifies Virtuozzo VE virtual configuration.

int modifyVirtuozzoVEConfig(VZLEnvCSP env)
{
    eid_t eid = env->getEID();
    VZAEEnvConfig envConfigOld = env->getVirtualConfig();
    VZAEEnvConfig envConfigNew;

    string sIn;
    cout << endl << "Modify configuration" << endl << endl;
    cout << "Current name: " << envConfigOld.getName().c_str() <<
endl;
    cout << "New Name: ";
    cin >> sIn;
    envConfigNew.setName(sIn);

    cout << endl;
    cout << "Current hostname: " << envConfigOld.getHostname().c_str()
<< endl;
    cout << "New Hostaname: ";
    cin >> sIn;
    envConfigNew.setHostname(sIn);
    cout << endl;

    VZLBoolOptionalProperty bOn;
    envConfigOld.getOnBoot(bOn);

```

```

    cout << "On-boot current: " << (bOn ? "on" : "off") << endl;
    cout << "On-boot (0 = off, 1 = on): ";
    cin >> sIn;
    envConfigNew.setOnBoot((sIn == "1" ? true : false));

    envConfigOld.getOfflineManagement(bOn);
    cout << "Offline mgmt current: " << (bOn ? "on" : "off") <<
endl;

    cout << "Offline management (0 = off, 1 = on): ";
    cin >> sIn;
    envConfigNew.setOfflineManagement(sIn == "1" ? true : false);

    // Create the Environment Management object using object factory.
    VZA::VZAEnvMSP vzaenvm = dynamic_pointer_cast<VZAEnvM>(
        VZLLibFunctionality::kit().getEnvM(
            accessPoint->getRootEID(),
            accessPoint,
            VZA::vzaFunctionalType));

    if (!vzaenvm)
    {
        cerr << "Error: unable to create VZA::VZAEnvMSP object." <<
endl;
        return -1;
    }

    if (vzaenvm->set(eid, envConfigNew) != 0)
    {
        VZLRequestErrorData errorData = getLastErrorData();
        cout << endl;
        cout << "Error (" << errorData.code << ") ";
        cout << errorData.message << endl;
        return -1;
    }

    cout << endl;
    return 0;
}

/*****/

// Displays a list of performance classes and
// counters and allows to make a selection.
int selectPerformanceCounters(string &classOut, string &counterOut)
{
    // Get the vocabulary from the root Environment.
    VZLVocMapSP vocabulary = accessPoint->getVocabulary(accessPoint-
>getRootEID());

    // List of categories.
    VZLCategoryRestrictSet categories;

    // Get the list of counters for Virtuozzo.
    // The restrictions on the resut set
    // are applied by using the insert() method and
    // passing the restriction type to it.
    // (remove or comment out the following line
    // to get counters for all environment types).
    categories.insert(vzaFunctionalType.c_str());

```

```

// Another restriction:
// getting only the vocabulary categories that
// themselves belong to the category
// "counters", i.e. the performance classes.
categories.insert("counters");

// Create an iterator to read the categories
// from the vocabulary.
auto_ptr<VZLVocMapIterator> vocIter(vocabulary-
>readCategory(categories));

// Check that vocabulary actually contains
// performance classes.
if (vocIter.get() == NULL || vocIter->firstCategory() != 0)
{
    return -1;
}

// Display the retrieved classes on the screen.
int ctr = 0;
string className;
std::vector<string> classIDList;
cout << "Performance Classes:" << endl << endl;
do
{
    vocIter->getID(className);
    classIDList.push_back(className);
    cout << "(" << ctr << " ) " << className << endl;
    ctr++;
} while (vocIter->nextCategory() == 0);

// Allow the user to select a class.
// This can be modified to allow selection
// of multiple classes. The code that follows
// must then be repeated in a loop for
// each selected class.
int classID;
cout << endl << "Enter class ID: ";
if (!(cin >> classID))
{
    cin.clear();
    cin.ignore(10000, '\n');
    cout << "Invalid class ID." << endl;
    return -1;
}

if (classID > int(classIDList.size()) || classID < 0)
{
    cout << "Invalid class ID." << endl;
    return -1;
}

cout << endl;

// Display a list of counters for the selected class.
auto_ptr<VZLVocMapIterator> cntrIter(vocabulary-
>readCategory(classIDList[classID]));

```



```

// Check that vocabulary contains counters for
// the specified class.
if (cntrIter.get() == NULL || cntrIter->firstParameter() != 0)
{
    cout << endl;
    cout << "No counters are available for the specified class.";
    cout << endl;
    return -1;
}

// A vector to store the counter names.
// Vector index will corresponde to the
// counter ID displayed on the screen to allow
// an easy selection of the counter.
std::vector<string> counterIDList;
ctr = 0;

// Get the list of counters for the selected class.
cout << "Counters: " << endl << endl;
do
{
    string counterName;
    int valueType, counterType;

    // Get counter information.
    cntrIter->getID(counterName);
    cntrIter->getValue(valueType, "value_type");
    cntrIter->getValue(counterType, "counter_type");

    // Insert the counter name into the "ID" vector.
    // Display counter information on the screen.
    counterIDList.push_back(counterName);
    cout << "(" << ctr << " ) " << counterName << " ( " <<
        ("valueType == 1" ? "integer" : "float") << ", " <<
        ("counterType == 1" ? "incremental" : "absolute") <<
        ")" << endl;
    ctr++;
} while (cntrIter->nextParameter() == 0);

// Allows the user to select the counter.
// This can be modified to select multiple counters.
int counterID;
cout << endl << "Enter counter ID: ";

if (!(cin >> counterID))
{
    cout << "Invalid counter ID." << endl;
    return -1;
}

if (counterID > int(counterIDList.size()) || counterID < 0)
{
    cin.clear();
    cin.ignore(10000, '\n');
    cout << "Invalid counter ID." << endl;
    return -1;
}

cout << endl;

```

```

        classOut = classIDList[classID];
        counterOut = counterIDList[counterID];

        return 0;
    }

    /***/

    // On-demand Performance Data.
    int getPerformanceReport(eid_t eid)
    {
        string className;
        string counterName;
        if (selectPerformanceCounters(className, counterName) != 0)
        {
            return -1;
        }

        // Store the selected classes and counters in the container.
        // This will be passed later to the method that will
        // retrieve the performance data.
        // If there are multiple instances of the class in your
        // system (e.g. multiple network interfaces), the name of
        // the instance must be used instead of the empty string.
        // To retrieve the data for multiple counters, the name of
        // each counter must be inserted into the container.
        VZLPerfMonClassListSP classList(new VZLPerfMonClassList);
        std::string instanceName = "";

        ((*classList)[classIDList[classID]][instanceName].insert(counterIDList[
counterID]));
        ((*classList)[className][instanceName].insert(counterName));

        // Initialize VZL plugin.
        initVZLFunctionalityAgent();

        // Create VZLPerfMon object.
        VZLPerfMonSP perfmon;
        perfmon = VZLLibFunctionality::kit().getPerfMon(
            accessPoint->getRootEID(),
            accessPoint);

        // The data will be retrieved for the specified EID,
        // the one that was passed to this function.
        // To retrieve the data for all known Environments,
        // leave the container empty.
        VZLEIDList eidList;
        eidList.insert(eid);

        // Get the performance data.
        VZLMonitorDataList monListOut;
        perfmon->get(&monListOut, eidList, classList, false);

        // The monListOut object should now contain the
        // requested performance data. Iterate through it and
        // display the data on the screen.
        // If multiple classes were selected, each iteration
        // displays the data for an individual class.
        for(VZLMonitorDataList::const_iterator it =
            monListOut.begin());

```

```

        it != monListOut.end(); ++it)
    {
        const VZLMonitorData& data = *it;

        cout << "Class:\t" << data.counterClass << endl;
        cout << "From:\t" << ctime(&data.interval.startTime);
        cout << "To:\t" << ctime(&data.interval.endTime);
        cout << endl;

        // This loop displays the data for each
        // class instance (if more than one instance exists).
        for(VZLMonitorData::InstanceMap::const_iterator
            inst = data.counters.begin();
            inst != data.counters.end(); ++inst)
        {
            cout << "Instance: " <<
                ("inst->first.length() == 0" ? "N/A" : inst->first)
                << endl << endl;

            // This loop displays the information for each
            // specified counter.
            for( VZLMonitorData::CountersMap::const_iterator counter =
                inst->second.begin();
                counter != inst->second.end(); ++counter )
            {
                cout << "Counter: " << counter->first << endl << endl;
                cout << "min\tmax\tavg\tcur" << endl;
                cout << string(30, '-') << endl;
                cout << counter->second.min.intval << "\t";
                cout << counter->second.max.intval << "\t";
                cout << counter->second.avg.intval << "\t";
                cout << counter->second.cur.intval << endl << endl;
            }
        }
        cout << endl;
    }
    return 0;
}

/*****

int startPerfMon(PerfMonHandlerSP perfMonHandler, eid_t eid)
{
    initVZLFunctionalityAgent();

    // Create the monitor object.
    // The constructor accepts the environment ID to
    // collect the data on, and a smart pointer to
    // the access point object.
    VZLMonitorAgent perfMon(accessPoint->getRootEID(), accessPoint);

    // Specify the classes and the counters that you
    // would like to use.
    VZLPerfMonClassList classes;

    string className;
    string counterName;

    if (selectPerformanceCounters(className, counterName) != 0)
    {

```

```

        return -1;
    }

    string instanceName = "";
    classes[className][instanceName].insert(counterName);

    // Start the monitor.
    VZLEIDList eidList;
    eidList.insert(eid);

    cout << "Starting monitor..." << endl;
    perfMon.async(perfMonHandler)->start(eidList,
        classes, 10,
        VZLPerfMonEnvFilter());

    return 0;
}

/*****

int getAlertList()
{
    // Create the Alert Managment object.
    VZL::VZLAlertMSP vzlAlertm;
    vzlAlertm = VZLLibFunctionality::kit().getAlertM(
        accessPoint->getRootEID(),
        accessPoint);

    // Get current alerts for all known environments.
    VZLAlertList pOut;
    if (vzlAlertm->getAlerts(&pOut) != 0)
    {
        cout << "Error, unable to get alert data." << endl;
        return -1;
    }

    // Examine the results.
    VZLAlertDataSP alertData;
    for (unsigned int i = 0; i < pOut.size(); ++i)
    {
        // Getting a message containing the summary alert info.
        cout << endl;
        cout << "Message: " << pOut[i]->info.toString() << endl <<
endl;

        // Get the base alert information. This information is common
        // to all categories of the alerts.
        cout << "ALERT: " << endl;
        cout << "Generated by (EID):\t" << pOut[i]->eid.toString() <<
endl;

        cout << "Alert source:\t\t" << pOut[i]->source << endl;
        cout << "Alert category:\t\t" << pOut[i]->category << endl;

        // Get the alert level. The received alert data object
        // always typecasts to VZLAlertData (the base alert data
class).
        alertData = dynamic_pointer_cast<VZLAlertData>(pOut[i]->data);
        cout << "Environment ID:\t\t" << alertData->eid.toString() <<
endl;

        cout << "Alert level:\t\t";

```

```

switch (alertData->type)
{
    case VZLAlertData::AT_GREEN:
        cout << "Green alert";
        break;
    case VZLAlertData::AT_YELLOW:
        cout << "Yello alert";
        break;
    case VZLAlertData::AT_RED:
        cout << "Red alert";
        break;
    case VZLAlertData::AT_BLACK:
        cout << "Black alert";
        break;
    default:
        cout << "Unknown alert type";
        break;
}
cout << endl;

// The information specific to a particular category is
// retrieved by typecasting the received object to the
appropriate
// type. Let's say that we want to do that for the
// Resource Allocation alerts.
if (pOut[i]->category == "resource_alert")
{
    VZLResourceAlertDataSP resourceAlertData =
        dynamic_pointer_cast<VZLResourceAlertData>(pOut[i]-
>data);
    cout << "Counter class:\t\t" << resourceAlertData-
>counterClass << endl;

    // For optional values, you have to check if
    // the value was actually specified.
    if (resourceAlertData->instance.isSpecified())
    {
        cout << "Instance:\t\t" << resourceAlertData-
>instance.get() << endl;
    }

    cout << "Counter:\t\t" << resourceAlertData->counter <<
endl;

    // A resource allocation alert may have its values
specified as
    // integers or floats.
    if (resourceAlertData->bInteger)
    {
        cout << "Current Value:\t\t" << resourceAlertData-
>cur.intval << endl;
        cout << "Hard Limit:\t\t" << resourceAlertData-
>hard.intval << endl;
        if (resourceAlertData->soft.isSpecified())
        {
            cout << "Soft Limit:\t\t" <<
resourceAlertData->soft->intval << endl;
        }
    }
}

```

```
        else
        {
            cout << "Current Value:\t\t" << resourceAlertData-
>cur.floatval << endl;
            cout << "Hard Limit:\t\t" << resourceAlertData-
>hard.floatval << endl;
            if(resourceAlertData->soft.isSpecified())
            {
                cout << "Soft Limit:\t\t" << resourceAlertData-
>soft->floatval << endl;
            }
        }
    }
    return 0;
}
```

Index

1

- 1. Creating the Project • 8
- 1. Managing Files and Directories • 79
- 1. Retrieving a List of Performance Counters • 56

2

- 2. Connecting to VZAgent • 9
- 2. Getting a Single Performance Report • 61
- 2. Managing Operating System Services • 85

3

- 3. Getting a List of Environments • 11
- 3. Receiving Periodic Performance Reports • 63

4

- 4. Creating a New Environment • 21
- 4. Receiving System Event Notifications • 67

5

- 5. Getting a List of Current Alerts • 71
- 5. Starting, Stopping, Restarting an Environment. • 26

6

- 6. Getting Environment Configuration • 28
- 6. Receiving Alerts by Subscription • 74

7

- 7. Modifying Environment Configuration • 33

8

- 8. Destroying an Environment • 36

9

- 9. The Complete Program Code • 37

A

- Adding Environment Monitoring • 55
- Appendix A Sample Program • 89

C

- Creating Your First Client Program • 8

D

- Documentation Conventions • 4

F

- Feedback • 5

G

- General Conventions • 5
- Glossary • 6

P

- Preface • 4

R

- Request Redirection • 77

S

- Shell Prompts in Command Examples • 5

T

- Typographical Conventions • 4