Contents

Chapter 1 Introduction to the Programming API Reference Guide
1
  About the Programming API ............................................. 1
  Compatibility .............................................................. 2
  Using the Programming API Documentation .......................... 2
  The Document Set .......................................................... 2
  Using the Reference Guide ................................................. 2
  Installing the Programming API ......................................... 3

Chapter 2 General Functions .............................................. 5
  Vix_FreeBuffer ............................................................ 6
    Function Signature ...................................................... 6
    Parameters ............................................................... 6
    Results ................................................................. 6
    Notes ................................................................. 6
    Example ............................................................... 6
  Vix_GetErrorText .......................................................... 7
    Function Signature ...................................................... 7
    Parameters ............................................................... 7
    Results ................................................................. 7
    Notes ................................................................. 7
  Vix_GetHandleType ....................................................... 8
    Function Signature ...................................................... 8
    Parameters ............................................................... 8
    Results ................................................................. 8
    Notes ................................................................. 8
  Vix_GetProperties ........................................................ 9
    Function Signature ...................................................... 9
    Parameters ............................................................... 9
    Results ................................................................. 9
    Notes ................................................................. 9
Chapter 3 Operations on Host Machines

Example

Vix_GetPropertyType
Function Signature
Parameters
Results
Notes

Vix_PumpEvents
Function Signature
Parameters
Results
Notes

Vix_ReleaseHandle
Function Signature
Parameters
Results
Notes

VixHost_Connect
Function Signature
Parameters
Results
Notes
Example

VixHost_Disconnect
Function Signature
Parameters
Results
Notes
Example

VixHost_FindItems
Function Signature
Parameters
Results
Notes
Example

VixHost_RegisterVM
Function Signature
VixVM_GetRootSnapshot  ................................................................. 40
  Function Signature ................................................................. 40
  Parameters ................................................................. 40
  Results ................................................................. 40
  Notes ................................................................. 40
  Example ................................................................. 40
VixVM_InstallTools ............................................................... 43
  Function Signature ................................................................. 43
  Parameters ................................................................. 43
  Results ................................................................. 43
  Notes ................................................................. 43
  Example ................................................................. 43
VixVM_LoginInGuest ............................................................ 46
  Function Signature ................................................................. 46
  Parameters ................................................................. 46
  Results ................................................................. 46
  Notes ................................................................. 46
  Example ................................................................. 46
VixVM_Open ................................................................. 50
  Function Signature ................................................................. 50
  Parameters ................................................................. 50
  Results ................................................................. 50
  Notes ................................................................. 50
  Example ................................................................. 51
VixVM_PowerOff ................................................................. 52
  Function Signature ................................................................. 52
  Parameters ................................................................. 52
  Results ................................................................. 52
  Notes ................................................................. 52
  Example ................................................................. 52
VixVM_PowerOn ................................................................. 54
  Function Signature ................................................................. 54
  Parameters ................................................................. 54
  Results ................................................................. 54
  Notes ................................................................. 54
  Example ................................................................. 54
VixVM_RemoveSnapshot .......................................................... 56
  Function Signature ................................................................. 56
Parameters .......................................................... 56
Results .............................................................. 56
Notes ............................................................... 56
Example ............................................................ 56
VixVM_Reset .......................................................... 59
  Function Signature .............................................. 59
  Parameters ....................................................... 59
  Results .......................................................... 59
  Notes ............................................................. 59
VixVM_RevertToSnapshot ........................................ 60
  Function Signature .............................................. 60
  Parameters ....................................................... 60
  Results .......................................................... 60
  Notes ............................................................. 60
Example ............................................................ 60
VixVM_RunProgramInGuest ....................................... 63
  Function Signature .............................................. 63
  Parameters ....................................................... 63
  Results .......................................................... 63
  Notes ............................................................. 63
Example ............................................................ 64
VixVM_Suspend ..................................................... 67
  Function Signature .............................................. 67
  Parameters ....................................................... 67
  Results .......................................................... 67
  Notes ............................................................. 67
VixVM_UpgradeVirtualHardware ................................ 68
  Function Signature .............................................. 68
  Parameters ....................................................... 68
  Results .......................................................... 68
  Notes ............................................................. 68
Example ............................................................ 68
VixVM_WaitForToolsInGuest ..................................... 70
  Function Signature .............................................. 70
  Parameters ....................................................... 70
  Results .......................................................... 70
  Notes ............................................................. 70
Example ............................................................ 70
Chapter 5 Operations on Job Handles ............................... 73
  VixJob_CheckCompletion ........................................... 74
    Function Signature .............................................. 74
    Parameters ....................................................... 74
    Results .......................................................... 74
    Notes ............................................................ 74
    Example .......................................................... 74
  VixJob_GetError ..................................................... 76
    Function Signature .............................................. 76
    Parameters ....................................................... 76
    Results .......................................................... 76
    Notes ............................................................ 76
    Example .......................................................... 76
  VixJob_Wait .......................................................... 78
    Function Signature .............................................. 78
    Parameters ....................................................... 78
    Results .......................................................... 78
    Notes ............................................................ 78
    Example .......................................................... 78

Appendix A Handle Types ........................................... 81

Appendix B Properties ................................................. 83
  Virtual Machine Properties ..................................... 83
  Event Info Property .............................................. 83
  Job Properties .................................................... 83
  Snapshot Properties ............................................. 84

Appendix C Error Codes ............................................. 85

Appendix D The Simple Perl API ................................. 91
  About the Simple Perl API ...................................... 91
  Installing the Simple Perl API ................................ 91
    Perl 5.8.0 and Perl 5.8.1 ...................................... 92
  Simple Perl API Quick Reference Guide ....................... 92
    Name and Description .......................................... 92
    General Functions ............................................. 92
    FindRunningVMs ................................................. 92
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetErrorText</td>
<td>92</td>
</tr>
<tr>
<td>GetHandleType</td>
<td>92</td>
</tr>
<tr>
<td>GetProperties</td>
<td>92</td>
</tr>
<tr>
<td>HostConnect</td>
<td>93</td>
</tr>
<tr>
<td>HostDisconnect</td>
<td>93</td>
</tr>
<tr>
<td>RegisterVM</td>
<td>93</td>
</tr>
<tr>
<td>ReleaseHandle</td>
<td>93</td>
</tr>
<tr>
<td>UnregisterVM</td>
<td>93</td>
</tr>
<tr>
<td>Virtual Machines Functions</td>
<td>93</td>
</tr>
<tr>
<td>VMCopyFileFromGuestToHost</td>
<td>93</td>
</tr>
<tr>
<td>VMCopyFileFromHostToGuest</td>
<td>94</td>
</tr>
<tr>
<td>VMDelate</td>
<td>94</td>
</tr>
<tr>
<td>VMLoginInGuest</td>
<td>94</td>
</tr>
<tr>
<td>VMOpen</td>
<td>94</td>
</tr>
<tr>
<td>VMPowerOff</td>
<td>94</td>
</tr>
<tr>
<td>VMPowerOn</td>
<td>94</td>
</tr>
<tr>
<td>VMReset</td>
<td>95</td>
</tr>
<tr>
<td>VMPowerOff</td>
<td>95</td>
</tr>
<tr>
<td>VMPowerOn</td>
<td>95</td>
</tr>
<tr>
<td>VMRunProgramInGuest</td>
<td>95</td>
</tr>
<tr>
<td>VMSuspend</td>
<td>95</td>
</tr>
<tr>
<td>VMWaitForToolsInGuest</td>
<td>95</td>
</tr>
<tr>
<td>Virtual Machine Snapshot Functions</td>
<td>95</td>
</tr>
<tr>
<td>VMCreateSnapshot</td>
<td>95</td>
</tr>
<tr>
<td>VMGetNumRootSnapshots</td>
<td>95</td>
</tr>
<tr>
<td>VMGetRootSnapshot</td>
<td>96</td>
</tr>
<tr>
<td>VMInstallTools</td>
<td>96</td>
</tr>
<tr>
<td>VMRemoveSnapshot</td>
<td>96</td>
</tr>
<tr>
<td>VMRevertToSnapshot</td>
<td>96</td>
</tr>
<tr>
<td>VMUpgradeVirtualHardware</td>
<td>96</td>
</tr>
</tbody>
</table>

**Appendix E The vmrun Utility** .................................................. 97

**Command-Line Application: vmrun** .................................................. 97

**To start the vmrun application** ................................................. 97

**Authentication Flags** ............................................................... 97

**Commands and Options** ............................................................. 97

**Examples for vmrun** ..................................................................... 98

**Tips** ............................................................................................ 99

**Appendix F Revision History** ....................................................... 101
Index .......................................................... 103
CHAPTER 1  Introduction to the Programming API Reference Guide

The Programming API Reference Guide describes an API that allows users to automate virtual machine operations on VMware Server. This API does not apply to any other VMware products at this time.

The following topics are covered in this chapter:

- “About the Programming API” on page 1 describes the purpose for the Programming API.
- “Using the Programming API Documentation” on page 2 describes the Programming API document set.
- “Installing the Programming API” on page 3 explains the basic installation requirements for creating and running Programming API clients.

About the Programming API

The Programming API (known as “Vix”) is an API that lets users write scripts and programs to manipulate virtual machines. It is high-level, easy to use, and practical for both script developers and application programmers. The Programming API is designed for three kinds of users:

- **Technically Adventurous Users** – Often such a user is a corporation with dedicated IT personnel that build their own in-house tools.

- **Partners** – These are typically software tools vendors that use this Programming API to better integrate VMware products with their own products or to build management products specifically for virtual machines.

- **VMware Products** – VMware uses the Programming API in its own products as a general abstraction layer between core virtual machine processes and associated service processes.

The Programming API runs on the Microsoft Windows and Linux platforms. This release supports clients written in C.
Compatibility
This release of the Programming API is compatible with VMware Server 1.0.

Using the Programming API Documentation

The Document Set
The following documents describe how to use the Programming API to create clients that manage virtual machines and hosts:

- Programming API Reference Guide (this book)
- Programming API Programming Guide

Using the Reference Guide
This section explains how to use this reference guide. This reference guide describes the API function calls in the following chapters:

1. “General Functions” on page 5 describes general-purpose functions such as error handling.
2. “Operations on Host Machines” on page 15 describes functions that apply to host machines, such as shutting down a host.
3. “Operations on Virtual Machines” on page 25 describes functions that apply to virtual machines, including the guest operating system.
4. “Operations on Job Handles” on page 73 describes functions for working with job handles for asynchronous operations.

You can browse the function call descriptions in each of those chapters to get an idea of the functions available for each subject. At the beginning of each chapter is a list of the functions described in that chapter. Each function description includes:

- the function signature
- the parameters to the function
- the return value of the function
- any special usage notes appropriate for the function
- (for some functions) an example of how to call the function in a C client

When you know the name of the function you want to look up, use the index to locate the function description quickly.

In addition, this reference guide contains these appendix chapters:
“Handle Types” on page 81 lists the handle types used to identify Programming API objects. The appendix is a short list with a brief explanation of each type.

“Properties” on page 83 lists the properties belonging to each handle type. The appendix is organized by handle type so that you can browse for the properties belonging to a handle with which you are working.

“Error Codes” on page 85 lists the error codes that could be returned from function calls.

“The Simple Perl API” on page 91 describes the installation and usage of the Perl API.

“The vmrun Utility” on page 97 explains how to use the vmrun command-line utility. The appendix is in alphabetic order so you can conveniently find a specific error code to determine its meaning.

Installing the Programming API

For complete instructions on installing the Programming API on a client machine, refer to the Programming API Programming Guide.
CHAPTER 2  General Functions

This chapter describes the API specifications for general-purpose Programming API functions, such as those functions that deal with handle management and errors. The functions covered in this chapter are:

- “Vix_FreeBuffer” on page 6
- “Vix_GetErrorText” on page 7
- “Vix_GetHandleType” on page 8
- “Vix_GetProperties” on page 9
- “Vix_GetPropertyType” on page 11
- “Vix_PumpEvents” on page 12
- “Vix_ReleaseHandle” on page 13
Vix_FreeBuffer

Function Signature

```c
void Vix_FreeBuffer(void *p);
```

Parameters

- `p`: A string pointer returned by a call to Vix_GetProperties() or Vix_JobWait().

Results

None.

Notes

When Vix_GetProperties() or Vix_JobWait() returns a string property, it allocates a buffer for the string. Client applications are responsible for calling Vix_FreeBuffer() to free the string buffer when no longer needed.

If you pass a null pointer to Vix_FreeBuffer(), the function returns immediately.

Example

This example retrieves the path name and power state of a virtual machine. Vix_FreeBuffer() is used to free the path name storage.

```c
VixError err = VIX_OK;
VixHandle myVM = VIX_INVALID_HANDLE;
char *pathName;
int vmPowerState = 0;

// ...Open the virtual machine and get a handle...

err = Vix_GetProperties(myVM,
    VIX_VM_VMX_PATHNAME,
    &pathName,
    VIX_VM_POWER_STATE_PROPERTY,
    &vmPowerState,
    VIX_PROPERTY_NONE);

if (VIX_OK != err) {
    // Handle the error...
}
Vix_FreeBuffer(pathName);
```
**Vix_GetErrorText**

**Function Signature**

\[
\text{const char *} \\
\text{Vix_GetErrorText(VixError err,} \text{ const char *locale);}
\]

**Parameters**

<table>
<thead>
<tr>
<th>err</th>
<th>A Vix error code returned by any other Vix function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>Must be NULL.</td>
</tr>
</tbody>
</table>

**Results**

A human-readable string that describes the error.

**Notes**

- This translates an error code, which is an enumerated type, to a user-readable string.
- The returned string is a constant string. Do not try to free it by calling `Vix_FreeBuffer()`.
- The language code is not used, and must be `NULL`. Only U.S. English is supported in this release.
Vix_GetHandleType

Function Signature

VixHandleType
Vix_GetHandleType(VixHandle handle);

Parameters

| handle | Any handle returned by a Vix function. |

Results

An enumerated type that identifies what kind of handle this is.

Notes

See “Handle Types” on page 81 for the currently supported handle types.
Vix_GetProperties

Function Signature

VixError
Vix_GetProperties(VixHandle handle,
                 VixPropertyID firstPropertyID,
                 ...);

Parameters

<table>
<thead>
<tr>
<th>handle</th>
<th>Any handle returned by a Vix function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstPropertyID</td>
<td>A property ID. See below for valid values.</td>
</tr>
</tbody>
</table>

Results

VixError. This function returns VIX_OK if it succeeded. Otherwise, it returns an appropriate error described in “Error Codes” on page 85.

Notes

- This function allows you to get one or more properties from a handle. You may use this function on any type of handle, but only specific properties are defined for each handle. See “Properties” on page 83 for the list of properties and the handle types to which they apply.
- This procedure accepts a variable number of parameters, so you can retrieve any number of properties with a single call. The parameters must be in a series of pairs of property IDs and result pointers. Each result pointer will accept the value of the property identified by the property ID in the previous parameter. The type of the pointer depends on the type of the property. You end the variable list of parameters with a single ID value of VIX_PROPERTY_NONE.
- When Vix_GetProperties() returns an error, the values of the output parameters are indeterminate.
- If you retrieve a string property, the Programming API allocates space for that string. You are responsible for calling Vix_FreeBuffer() to free the string.

Example

VixError err = VIX_OK;
int vmPowerState;
char *vmPathName = NULL;
VixHandle myVM = VIX_INVALID_HANDLE;
// ...Open the virtual machine and get a handle...
err = Vix_GetProperties(myVM,
        VIX_PROPERTY_VM_POWER_STATE,
        &vmPowerState,
        VIX_PROPERTY_VM_VMX_PATHNAME,
        &vmPathName,
        VIX_PROPERTY_NONE);

if (VIX_OK != err) {
    // Handle the error...
}
Vix_FreeBuffer(vmPathName);
Vix_GetPropertyType

Function Signature

VixError
Vix_GetPropertyType(VixHandle handle,
VixPropertyID PropertyID,
VixPropertyType *propertyType);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>Any handle returned by a Vix function.</td>
</tr>
<tr>
<td>propertyID</td>
<td>A property ID. See below for valid values.</td>
</tr>
<tr>
<td>propertyType</td>
<td>The type of the data stored by the property.</td>
</tr>
</tbody>
</table>

Results

VixError. This function returns VIX_OK if it succeeded. Otherwise, it returns an appropriate error described in “Error Codes” on page 85.

Notes

This function identifies the data type of a property. Possible values are:

- integer
- string
- bool
- handle
- int64
Vix_PumpEvents

Function Signature

```c
void
Vix_PumpEvents(VixHandle hostHandle,
                 VixPumpEventsOptions options);
```

Parameters

<table>
<thead>
<tr>
<th>handle</th>
<th>The handle to the local host object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Must be 0.</td>
</tr>
</tbody>
</table>

Results

None.

Notes

- This option is designed for client applications that want to receive events on only one thread, such as a thread that manages the user interface. In this case, open the local host handle with the VIX_HOSTOPTION_USE_EVENT_PUMP option, and call Vix_PumpEvents() in the main event loop. If you fail to call Vix_PumpEvents(), Vix asynchronous actions will not execute.

- If you do not open your local host with the VIX_HOSTOPTION_USE_EVENT_PUMP option, calling Vix_PumpEvents() has no effect.

If you do not open your local host with the VIX_HOSTOPTION_USE_EVENT_PUMP option, the Programming API creates a small pool of threads for executing asynchronous operations. Events can be posted from those worker threads, so your code must be thread-safe.
**Vix_ReleaseHandle**

**Function Signature**

```c
void
Vix_ReleaseHandle(VixHandle handle);
```

**Parameters**

| handle       | Any handle returned by a Vix function. |

**Results**

None.

**Notes**

This function decrements the reference count for a handle. You should no longer use the handle once it has been released. When the last reference to a handle has been released, the handle’s runtime state is destroyed.
This chapter describes the API specifications for Programming API functions related to
host machines, such as those functions that deal with stopping a local host or counting
the virtual machines running on the host. The functions covered in this chapter are:

- “VixHost_Connect” on page 16
- “VixHost_Disconnect” on page 18
- “VixHost_FindItems” on page 20
- “VixHost_RegisterVM” on page 23
- “VixHost_UnregisterVM” on page 24
VixHost_Connect

Function Signature

VixHandle
VixHost_Connect(int apiVersion,
VixServiceProvider hostType,
const char *hostName,
int hostPort,
const char *userName,
const char *password,
VixHostOptions options,
VIXHandle propertyListHandle,
VIXEventProc *callbackProc,
void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiVersion</td>
<td>The version of the API to be supported. This is defined as VIX_API_VERSION.</td>
</tr>
<tr>
<td>hostType</td>
<td>The type of host being opened. Must be VIX_SERVICEPROVIDER_VMWARE_SERVER.</td>
</tr>
<tr>
<td>hostName</td>
<td>Name of the host to which you want to connect.</td>
</tr>
<tr>
<td>hostPort</td>
<td>Port number on the host to which you want to connect.</td>
</tr>
<tr>
<td>userName</td>
<td>Name of user for authentication on the host.</td>
</tr>
<tr>
<td>password</td>
<td>Password of user for authentication on the host.</td>
</tr>
<tr>
<td>options</td>
<td>Options for the host. See the Notes section below.</td>
</tr>
<tr>
<td>propertyListHandle</td>
<td>Must be VIX_INVALID_HANDLE.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback that will be invoked when this function completes.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous call.

Notes

- This function initializes the host object. You must call this before calling any other Vix function. The host object is used for all local Vix operations.
- This function is asynchronous. It uses a job object to report when the operation is complete. The function returns a handle to the job object immediately. When the job is signaled, the host handle is stored as the VIX_PROPERTY_JOB_RESULT_HANDLE property of the job object.
To specify the local host (where the API client runs), pass null values for the
hostName, hostPort, userName, and password parameters. This is shown in the
element below.

If you are already connected to the host, a subsequent call to VixHost_Connect()
succeeds if you connect as the same user and use the same host name. Subsequent
calls return the same handle value.

When you initialize the host object, you can also control some Vix operations with
the options parameter. The following option is supported:

VIX_HOSTOPTION_USE_EVENT_PUMP — All asynchronous event processing
happens when the client calls Vix_PumpEvents(). The client is responsible for
regularly calling Vix_PumpEvents(), such as in an event loop.

Example

```c
int main()
{
    VixHandle hostHandle = Vix_INVALID_HANDLE;
    VixHandle jobHandle = Vix_INVALID_HANDLE;
    VixError err;
    jobHandle = VixHost_Connect(Vix_API_VERSION,
        VIX_SERVICEPROVIDER_VMWARE_SERVER,
        NULL, // hostName
        0, // hostPort
        NULL, // userName
        NULL, // password,
        0, // options
        Vix_INVALID_HANDLE, // propertyListHandle
        NULL, // callbackProc
        NULL); // clientData
    err = VixJob_Wait(jobHandle,
        VIX_PROPERTY_JOB_RESULT_HANDLE,
        &hostHandle,
        VIX_PROPERTY_NONE);
    if (Vix_OK != err) {
        // Handle the error...
        goto abort;
    }
    printf("Connected to local host.\n");
    abort;
    Vix_ReleaseHandle(jobHandle);
    VixHost_Disconnect(hostHandle);
}
```
VixHost_Disconnect

Function Signature

```c
void
VixHost_Disconnect(VixHandle hostHandle);
```

Parameters

| handle | The host handle returned by VixHost_Connect(). |

Results

None.

Notes

Call this function to disconnect the host. After you call this function the handle is no longer valid and you should not use it in any Vix function. Similarly, you should not use any handles obtained from the host while it was connected.

Example

```c
int main()
{
    VixHandle hostHandle = Vix_INVALID_HANDLE;
    VixHandle jobHandle = Vix_INVALID_HANDLE;
    VixError err;
    jobHandle = VixHost_Connect(Vix_API_VERSION,
                                VIX_SERVICEPROVIDER_VMWARE_SERVER,
                                NULL, // hostName
                                0, // hostPort
                                NULL, // userName
                                NULL, // password,
                                0, // options
                                Vix_INVALID_HANDLE, // propertyListHandle
                                NULL, // callbackProc
                                NULL); // clientData
    err = VixJob_Wait(jobHandle,
                      VIX_PROPERTY_JOB_RESULT_HANDLE,
                      &hostHandle,
                      VIX_PROPERTY_NONE);
    if (Vix_OK != err) {
        // Handle the error...
    }
    // Release the job handle because we no longer need it.
    Vix_ReleaseHandle(jobHandle);
```
// ...Do everything in your program...
VixHost_Disconnect(hostHandle);
return (0);
}
**VixHost_FindItems**

**Function Signature**

```c
VixHandle VixHost_FindItems(VixHandle hostHandle,
                             VixFindItemType searchType,
                             VixHandle searchCriteria,
                             int32 timeout,
                             VixEventProc *callbackProc,
                             void *clientData);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostHandle</td>
<td>The host handle returned by VixHost_Connect().</td>
</tr>
<tr>
<td>searchType</td>
<td>The type of items to find. Must be Vix_FIND_RUNNING_VMS in this release.</td>
</tr>
<tr>
<td>searchCriteria</td>
<td>Must be VIX_INVALID_HANDLE.</td>
</tr>
<tr>
<td>timeout</td>
<td>Must be -1.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A function to be invoked when VixHost_FindItems() completes.</td>
</tr>
<tr>
<td>clientData</td>
<td>A user-supplied parameter to be passed to the callback function.</td>
</tr>
</tbody>
</table>

**Results**

VixHandle. A job handle that describes the state of this asynchronous call.

**Notes**

This function asynchronously finds Vix objects and calls the application's callback function to report each object found. For example, when used to find all running virtual machines, VixHost_FindItems() returns a series of virtual machine file path names.

Each time VixHost_FindItems() calls the callback function, it passes the following things to the callback function:

- When an object is found, VixHost_FindItems() passes the value Vix_EVENTTYPE_FIND_ITEM in the eventType parameter.
- When an object is found, VixHost_FindItems() passes a property list object containing the location of the object found. The property list is passed as the value of the moreEventInfo parameter. The property list contains a single string value (Vix_PROPERTY_FOUND_ITEM_LOCATION), which is the URL of the item found.
- When VixHost_FindItems() is done searching, it calls the callback function a final time and passes Vix_EVENTTYPE_CALLBACK_SIGNALLED as the event type.
Example

The following example prints the path name of every virtual machine currently running on the host.

```c
static VixHandle hostHandle = Vix_INVALID_HANDLE;

void VixDiscoveryProc(VixHandle jobHandle,
    VixEventType eventType,
    VixHandle moreEventInfo,
    void *clientData)
{
    VixError err = Vix_OK;
    VixHandle jobHandle = Vix_INVALID_HANDLE;
    VixHandle vmHandle = Vix_INVALID_HANDLE;
    char *url = NULL;

    // Check callback event; ignore progress reports.
    if (Vix_EVENTTYPE_FIND_ITEM != eventType) {
        return;
    }

    // Found a virtual machine.
    err = Vix_GetProperties(moreEventInfo,
        Vix_PROPERTY_FOUND_ITEM_LOCATION,
        &url,
        VIX_PROPERTY_NONE);

    if (Vix_OK != err) {
        // Handle the error...
        goto abort;
    }

    printf("Found virtual machine: %s", url);

    abort:
    Vix_ReleaseHandle(jobHandle);
    Vix_FreeBuffer(url);
}

int main()
{
    VixHandle jobHandle = Vix_INVALID_HANDLE;
    VixHandle hostHandle = Vix_INVALID_HANDLE;
    VixError err;

    jobHandle = VixHost_Connect(Vix_API_VERSION,
        VIX_SERVICEPROVIDER_VMWARE_SERVER,
        NULL, // hostName
        0, // hostPort
        NULL, // userName
        NULL, // password,
        0, // options

Vix_INVALID_HANDLE, // propertyListHandle
NULL, // callbackProc
NULL); // clientData

err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &hostHandle,
    VIX_PROPERTY_NONE);

if (Vix_OK != err) {
    // Handle the error...
    goto abort;
}

// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

printf("Looking for running virtual machines");
jobHandle = VixHost_FindItems(hostHandle,
    Vix_FIND_RUNNING_VMS,
    VIX_INVALID_HANDLE, // searchCriteria
    -1, // timeout
    VixDiscoveryProc,
    NULL);

VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

abort:
    Vix_ReleaseHandle(jobHandle);
    VixHost_Disconnect(hostHandle);
}
Chapter 3 Operations on Host Machines

VixHost_RegisterVM

Function Signature

VixHandle
VixHost_RegisterVM(VixHandle hostHandle,
                 const char *vmxFilePath,
                 VixEventProc *callbackProc,
                 void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostHandle</td>
<td>The host handle returned by VixHost_Connect().</td>
</tr>
<tr>
<td>vmxFilePath</td>
<td>The path name of the .vmx file on the local host.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A function to be invoked when VixHost_UnregisterVM() completes.</td>
</tr>
<tr>
<td>clientData</td>
<td>A user-supplied parameter to be passed to the callback function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous call.

Notes

- This function adds a virtual machine to the local host's inventory. The virtual machine is identified by the vmxFilePathName, which is a path name to the configuration file (.VMX file) for that virtual machine.

- The format of the path name depends on the host operating system. If the path name includes backslash characters, you need to precede each one with an escape character.

- This function is asynchronous. Completion is reported by a job handle.
VixHost_UnregisterVM

Function Signature

VixHandle
VixHost_UnregisterVM(VixHandle hostHandle,
const char *vmxFilePath,
VixEventProc *callbackProc,
void *clientData);

Parameters

<table>
<thead>
<tr>
<th>hostHandle</th>
<th>The host handle returned by VixHost_Connect().</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmxFilePath</td>
<td>The path name of the .vmx file on the local host.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A function to be invoked when VixHost_UnregisterVM() completes.</td>
</tr>
<tr>
<td>clientData</td>
<td>A user-supplied parameter to be passed to the callback function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous call.

Notes

- This function removes a virtual machine from the local host's inventory. The virtual machine is identified by the vmxFilePathName, which is a path name to the configuration file (.VMX file) for that virtual machine.
- The format of the path name depends on the host operating system. If the path name includes backslash characters, you need to precede each one with an escape character.
- This function is asynchronous. Completion is reported by a job handle.
CHAPTER 4  Operations on Virtual Machines

This chapter describes the API specifications for Programming API functions that are related to virtual machines, such as those functions that deal with starting a virtual machine and installing VMware Tools. The functions covered in this chapter are:

- “VixVM_CopyFileFromGuestToHost” on page 26
- “VixVM_CopyFileFromHostToGuest” on page 30
- “VixVM_CreateSnapshot” on page 34
- “VixVM_Delete” on page 37
- “VixVM_GetNumRootSnapshots” on page 39
- “VixVM_GetRootSnapshot” on page 40
- “VixVM_InstallTools” on page 43
- “VixVM_LoginInGuest” on page 46
- “VixVM_Open” on page 50
- “VixVM_PowerOff” on page 52
- “VixVM_PowerOn” on page 54
- “VixVM_RemoveSnapshot” on page 56
- “VixVM_Reset” on page 59
- “VixVM_RevertToSnapshot” on page 60
- “VixVM_RunProgramInGuest” on page 63
- “VixVM_Suspend” on page 67
- “VixVM_UpgradeVirtualHardware” on page 68
- “VixVM_WaitForToolsInGuest” on page 70
VixVM_CopyFileFromGuestToHost

Function Signature

VixHandle
VixVM_CopyFileFromGuestToHost(VixHandle vmHandle,
const char* guestPathName,
const char* hostPathName,
int options,
VixHandle propertyListHandle,
VixEventProc *callbackProc,
void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>guestPathName</td>
<td>The path name of a file on a file system available to the guest.</td>
</tr>
<tr>
<td>hostPathName</td>
<td>The path name of a file on a file system available to the host.</td>
</tr>
<tr>
<td>options</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>propertyList</td>
<td>Must be VIX_INVALID_HANDLE.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

- This function copies a file from the guest operating system to the host operating system. The virtual machine must be running while the file is copied.
- The format of the file name depends on the guest or host operating system. For example, a path name for a Microsoft Windows guest or host requires backslash as a directory separator, whereas a Linux guest or host requires a forward slash. If the path name includes backslash characters, you need to precede each one with an escape character.
- You must call VixVM_LoginInGuest() before calling this procedure.
The copy operation requires VMware Tools to be installed and running in the guest operating system.

If any file fails to be copied, the operation returns an error. In this case, Vix aborts the operation and does not attempt to copy the remaining files.

Example

This example copies a compiled object file from a virtual machine to be run on the host.

```c
VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
   VIX_SERVICEPROVIDER_VMWARE_SERVER,
   NULL, // hostName
   0, // hostPort
   NULL, // userName
   NULL, // password
   0, // options
   VIX_INVALID_HANDLE, // propertyListHandle
   NULL, // callbackProc
   NULL); // clientData

err = VixJob_Wait(jobHandle,
   VIXPROPERTY_JOB_RESULT_HANDLE,
   &hostHandle,
   VIXPROPERTY_NONE);
if (VIX_OK != err) {
   // Handle the error...
   goto abort;
}

// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
   "c:\Virtual Machines\vm1\win2000.vmx",
   NULL, // callbackProc
   NULL); // clientData

err = VixJob_Wait(jobHandle,
   VIXPROPERTY_JOB_RESULT_HANDLE,
   &vmHandle,
   VIXPROPERTY_NONE);
if (VIX_OK != err) {
   // Handle the error...
   goto abort;
}

// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);
```
// Power on the virtual machine before copying file.
jobHandle = VixVM_PowerOn(vmHandle,
    0, // powerOnOptions
    Vix_INVALID_HANDLE, // propertyListHandle
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Wait until guest is completely booted.
jobHandle = VixVM_WaitForToolsInGuest(vmHandle,
    300, // timeoutInSeconds
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Authenticate for guest operations.
jobHandle = VixVM_LoginInGuest(vmHandle,
    "vixuser", // userName
    "secret", // password
    0, // options
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Copy the file.
jobHandle = VixVM_CopyFileFromGuestToHost(vmHandle,
    "c:\guestDir\helloworld.o", // src name
    "c:\hostDir\helloworld.o", // dest name
    0, // options
    Vix_INVALID_HANDLE, // propertyListHandle
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle, VIXPROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
VixVM_CopyFileFromHostToGuest

Function Signature

VixHandle VixVM_CopyFileFromHostToGuest(VixHandle vmHandle, const char* hostPathName, const char* guestPathName, int options, VixHandle propertyListHandle, VixEventProc *callbackProc, void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>hostPathName</td>
<td>The path name of a file on a file system available to the host.</td>
</tr>
<tr>
<td>guestPathName</td>
<td>The path name of a file on a file system available to the guest.</td>
</tr>
<tr>
<td>options</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>propertyList</td>
<td>Must be VIX_INVALID_HANDLE.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

- This function copies a file from the host operating system to the guest operating system. The virtual machine must be running while the file is copied.
- The format of the file name depends on the guest or host operating system. For example, a path name for a Microsoft Windows guest or host requires backslash as a directory separator, whereas a Linux guest or host requires a forward slash. If the path name includes backslash characters, you need to precede each one with an escape character.
- You must call VixVM_LoginInGuest() before calling this procedure.
- The copy operation requires VMware Tools to be installed and running in the guest operating system.
If any file fails to be copied, the operation returns an error. In this case, Vix aborts the operation and does not attempt to copy the remaining files.

Example

This example copies a source file from the host so it can be compiled in a virtual machine.

```c
VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
  VIX_SERVICEPROVIDER_VMWARE_SERVER,
  NULL, // hostName
  0, // hostPort
  NULL, // userName
  NULL, // password,
  0, // options
  VIX_INVALID_HANDLE, // propertyListHandle
  NULL, // callbackProc
  NULL); // clientData

err = VixJob_Wait(jobHandle,
  VIX_PROPERTY_JOB_RESULT_HANDLE,
  &hostHandle,
  VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
  "c:\Virtual Machines\vm1\win2000.vmx",
  NULL, // callbackProc
  NULL); // clientData

err = VixJob_Wait(jobHandle,
  VIX_PROPERTY_JOB_RESULT_HANDLE,
  &vmHandle,
  VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Power on the virtual machine before copying file.
```
jobHandle = VixVM_PowerOn(vmHandle,
    0, // powerOnOptions
    Vix_INVALID_HANDLE, // propertyListHandle
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle,VIXPROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Wait until guest is completely booted.
jobHandle = VixVM_WaitForToolsInGuest(vmHandle,
    300, // timeoutInSeconds
    NULL, // callbackProc,
    NULL); // clientData
err = VixJob_Wait(jobHandle, VIXPROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Authenticate for guest operations.
jobHandle = VixVM_LoginInGuest(vmHandle,
    "vixuser", // userName
    "secret", // password
    0, // options
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle, VIXPROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Copy a file.
jobHandle = VixVM_CopyFileFromHostToGuest(vmHandle,
    "c:\hostDir\helloworld.c", // src name
    "c:\guestDir\helloworld.c", // dest name
    0, // options
    Vix_INVALID_HANDLE, // propertyList
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle, VIXPROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
VixVM_CreateSnapshot

Function Signature

VixHandle
VixVM_CreateSnapshot(VixHandle vmHandle,
const char *name,
const char *description,
int options,
VixHandle propertyListHandle,
VixEventProc *callbackProc,
void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>name</td>
<td>Must be NULL.</td>
</tr>
<tr>
<td>description</td>
<td>Must be NULL.</td>
</tr>
<tr>
<td>options</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>propertyListHandle</td>
<td>Must be VIX_INVALID_HANDLE.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

- This function saves a copy of the virtual machine state as a snapshot object. The handle of the snapshot object is returned in the job object properties.
- This release of the Programming API supports only a single snapshot for each virtual machine. If you call this function a second time for the same virtual machine without first deleting the snapshot, the second call will overwrite the previous snapshot.

A virtual machine imported from another VMware product might have more than one snapshot at the time it is imported. In that case, you can use this function to add a new snapshot to the series.
Example

VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
VixHandle snapshotHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
                       VIX_SERVICEPROVIDER_VMWARE_SERVER,
                       NULL, // hostName
                       0, // hostPort
                       NULL, // userName
                       NULL, // password
                       0, // options
                       VIX_INVALID_HANDLE, // propertyListHandle
                       NULL, // callbackProc
                       NULL); // clientData

err = VixJob_Wait(jobHandle,
                VIX_PROPERTY_JOB_RESULT_HANDLE,
                &hostHandle,
                VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
                       "c:\\Virtual Machines\\vm1\\win2000.vmx",
                       NULL, // callbackProc
                       NULL); // clientData
err = VixJob_Wait(jobHandle,
                VIX_PROPERTY_JOB_RESULT_HANDLE,
                &vmHandle,
                VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Power on the virtual machine before creating snapshot.
jobHandle = VixVM_PowerOn(vmHandle,
                           0, // powerOnOptions
                           VIX_INVALID_HANDLE, // propertyListHandle
                           NULL, // callbackProc
                           NULL); // clientData
err = VixJob_Wait(jobHandle,VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Create snapshot in powered-on state.
jobHandle = VixVM_CreateSnapshot(vmHandle,
    NULL, // name
    NULL, // description
    0, // options
    Vix_INVALID_HANDLE, // propertyListHandle
    NULL, // callbackProc
    NULL); // clientData

err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &snapshotHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(snapshotHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
**VixVM_Delete**

**Function Signature**

```c
VixHandle VixVM_Delete(VixHandle vmHandle,
VixVMDeleteOptions deleteOptions,
VixEventProc *callbackProc,
void *clientData);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call <code>VixVM_Open</code> to create a virtual machine handle.</td>
</tr>
<tr>
<td>deleteOptions</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the operation completes.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the <code>callbackProc</code> function.</td>
</tr>
</tbody>
</table>

**Results**

VixHandle. A job handle that describes the state of this asynchronous call.

**Notes**

- This function permanently deletes a virtual machine from your host system. You can accomplish the same effect by deleting all virtual machine files using the host's file system. This function simplifies the task by deleting all VMware files in a single function call. This function does not delete other user files in the virtual machine folder.
- This function is successful only if the virtual machine is powered off or suspended.
- This function is asynchronous. It uses a job handle to report when it is complete.

**Example**

```c
VixError err = Vix_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
    VIX_SERVICEPROVIDER_VMWARE_SERVER,
    NULL, // hostName
    0, // hostPort
    NULL, // userName
```
NULL, // password,
0, // options
VIX_INVALID_HANDLE, // propertyListHandle
NULL, // callbackProc
NULL); // clientData

err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &hostHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
    "c:\Virtual Machines\vm1\win2000.vmx",
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &vmHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Delete(vmHandle, 0, NULL, NULL);
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

abort:
    Vix_ReleaseHandle(jobHandle);
    Vix_ReleaseHandle(vmHandle);
    VixHost_Disconnect(hostHandle);
VixVM_GetNumRootSnapshots

Function Signature

VixError
VixVM_GetNumRootSnapshots(VixHandle vmHandle,
    int *result);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>result</td>
<td>The number of root snapshots on this virtual machine.</td>
</tr>
</tbody>
</table>

Results

VixError. Refer to “Error Codes” on page 85 for possible error values.

Notes

This function returns the number of top-level (root) snapshots belonging to a virtual machine. A top-level snapshot is one that is not based on any previous snapshot. If the virtual machine has more than one snapshot, the snapshots can be a sequence in which each snapshot is based on the previous one, leaving only a single top-level snapshot. However, if applications create branched trees of snapshots, a single virtual machine can have several top-level snapshots.

This release of the Programming API can manage only a single snapshot for each virtual machine. All other snapshots in a sequence are ignored. The value of the result parameter is always 0 or 1.
VixVM_GetRootSnapshot

Function Signature

VixError
VixVM_GetRootSnapshot(VixHandle vmHandle,
                        int index,
                        VixHandle *snapshotHandle);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>index</td>
<td>Identifies a root snapshot. See below for range of values.</td>
</tr>
<tr>
<td>snapshotHandle</td>
<td>An output parameter that receives a handle to a snapshot.</td>
</tr>
</tbody>
</table>

Results

VixError. Refer to “Error Codes” on page 85 for possible error values.

Notes

- This function returns the handle of the specified snapshot belonging to the virtual machine referenced by vmHandle.
- Snapshots are indexed from 0 to n-1, where n is the number of root snapshots. Use the function VixVM_GetNumRootSnapshots() to get the value of n.
- This release of the Programming API can manage only a single snapshot for each virtual machine. The value of index can only be 0.
- The snapshotHandle returned by this function is reference counted. The calling application is responsible for releasing the handle.

Example

VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
VixHandle snapshotHandle = VIX_INVALID_HANDLE;
int numRootSnapshots;
int index;

jobHandle = VixHost_Connect(VIX_API_VERSION,
                            VIX_SERVICEPROVIDER_VMWARE_SERVER,
                            NULL, // hostName
                            ...);
Chapter 4 Operations on Virtual Machines

0, // hostPort
NULL, // userName
NULL, // password,
0, // options
VIX_INVALID_HANDLE, // propertyListHandle
NULL, // callbackProc
NULL); // clientData

err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &hostHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
    "c:\Virtual Machines\vm1\win2000.vmx",
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &vmHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// No need to release jobHandle here. It’s released at bottom.

// Only 1 snapshot supported in this release.
numRootSnapshots = 1;
for (index = 0; index < numRootSnapshots; index++) {
    err = Vix_VM_GetNumRootSnapshots(vmHandle, &numRootSnapshots);
    if (VIX_OK != err) {
        // Handle the error...
        goto abort;
    }
    for (index = 0; index < numRootSnapshots; index++) {
        err = VixVM_GetRootSnapshot(vmHandle,
            index,
            &snapshotHandle);
        if (VIX_OK != err) {
            // Handle the error...
            goto abort;
        }
        // ...Access snapshot properties...
        // Release the snapshot handle because we no longer need it.
Vix_ReleaseHandle(snapshotHandle);
}

abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(snapshotHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
**VixVM_InstallTools**

**Function Signature**

```c
VixHandle VixVM_InstallTools(VixHandle vmHandle,
int options,
char *commandLineArgs,
VixEventProc *callbackProc,
void *clientData);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>options</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>commandLineArgs</td>
<td>Must be NULL.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

**Results**

VixHandle. A job handle that describes the state of this asynchronous operation.

**Notes**

- Installs VMware Tools on the guest operating system. If VMware Tools is already installed, this function upgrades VMware Tools to the version that matches the Vix library.
- The virtual machine must be powered on to do this operation.

**Example**

```c
VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
    VIX_SERVICEPROVIDER_VMWARE_SERVER,
    NULL, // hostName
    0, // hostPort
    NULL, // userName
```
NULL, // password,
0, // options
VIX_INVALID_HANDLE, // propertyListHandle
NULL, // callbackProc
NULL); // clientData

err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &hostHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
    "c:\Virtual Machines\vm1\win2000.vmx",
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &vmHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_PowerOn(vmHandle,
    0, // powerOnOptions
    Vix_UI_NONE,
    Vix_NULL_HANDLE, // propertyListHandle
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Install VMware Tools in running virtual machine.
jobHandle = VixVM_InstallTools(vmHandle,
    0, // options
    NULL, // commandLineArgs
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_PowerOff(vmHandle,
0, // powerOffOptions
NULL, // callbackProc
NULL); // clientData
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
VixVM_LoginInGuest

Function Signature

VixHandle VixVM_LoginInGuest(VixHandle vmHandle, char *userName, char *password, int options, VixEventProc *callbackProc, void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of a user account on the guest operating system.</td>
</tr>
<tr>
<td>password</td>
<td>The password of the account identified by userName.</td>
</tr>
<tr>
<td>options</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

- This function validates the account name and password. You must call this function before calling functions to perform operations on the guest operating system, such as VixVM_RunProgramInGuest(). If you do not call any guest functions, you do not need to call VixVM_LoginInGuest().
- The virtual machine must be powered on before calling this function.
- VMware Tools must be installed and running on the guest operating system before calling this function.

Example

This example copies a compiled object file from a virtual machine to be run on the host. As described in “VixVM_CopyFileFromGuestToHost” on page 26, you must call VixVM_LoginInGuest() before calling the copy procedure.
VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
VIX_SERVICEPROVIDER_VMWARE_SERVER,
NULL, // hostName
0, // hostPort
NULL, // userName
NULL, // password,
0, // options
VIX_INVALID_HANDLE, // propertyListHandle
NULL, // callbackProc
NULL); // clientData

err = VixJob_Wait(jobHandle,
VIX_PROPERTY_JOB_RESULT_HANDLE,
&hostHandle,
VIX_PROPERTY_NONE);
if (VIX_OK != err) {
// Handle the error...
go to abort;
}

// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
"c:\Virtual Machines\vm1\win2000.vmx",
NULL, // callbackProc
NULL); // clientData
err = VixJob_Wait(jobHandle,
VIX_PROPERTY_JOB_RESULT_HANDLE,
&vmHandle,
VIX_PROPERTY_NONE);
if (VIX_OK != err) {
// Handle the error...
go to abort;
}

// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Power on the virtual machine before copying file.
jobHandle = VixVM_PowerOn(vmHandle,
0, // powerOnOptions
VIX_INVALID_HANDLE, // propertyListHandle
NULL, // callbackProc
NULL); // clientData
err = VixJob_Wait(jobHandle,VIX_PROPERTY_NONE);
if (VIX_OK != err) {
// Handle the error...
goto abort;

// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Wait until guest is completely booted.
jobHandle = VixVM_WaitForToolsInGuest(vmHandle,
    300, // timeoutInSeconds
    NULL, // callbackProc,
    NULL); // clientData

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Authenticate for guest operations.
jobHandle = VixVM_LoginInGuest(vmHandle,
    "vixuser", // userName
    "secret", // password
    0, // options
    NULL, // callbackProc
    NULL); // clientData

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Copy the file.
jobHandle = VixVM_CopyFileFromGuestToHost(vmHandle,
    "c:\guestDir\helloworld.o", // src name
    "c:\\hostDir\\helloworld.o", // dest name
    0, // options
    Vix_INVALID_HANDLE, // propertyList
    NULL, // callbackProc
    NULL); // clientData

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
VixVM_Open

Function Signature

VixHandle
VixVM_Open(VixHandle hostHandle,
const char *vmxFilePathName,
VixEventProc *callbackProc,
void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostHandle</td>
<td>The handle of a host object, typically returned from VixHost_Connect().</td>
</tr>
<tr>
<td>vmxFilePathName</td>
<td>The path name of the virtual machine configuration file on the local host.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when VixVM_Open() completes.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc procedure.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous call.

Notes

- This function opens a virtual machine on the host that is identified by the hostHandle parameter. The virtual machine is identified by vmxFilePathName, which is a path name to the configuration file (.VMX file) for that virtual machine.

- The format of the path name depends on the host operating system. For example, a path name for a Windows host requires backslash as a directory separator, whereas a Linux host requires a forward slash. If the path name includes backslash characters, you need to precede each one with an escape character.

- This function is asynchronous, and uses a job object to report when the operation is complete. The function returns a handle to the job object immediately. When the job is signaled, the virtual machine handle is stored as the VIX_PROPERTY_JOB_RESULT_HANDLE property of the job object.

- A virtual machine must be registered before you can open it. You can register a virtual machine by opening it with the VMware Server Console or by using the following command in a shell window:

  vmware-cmd register <path to .vmx file>
Example

```c
VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
    VIX_SERVICEPROVIDER_VMWARE_SERVER,
    NULL, // hostName
    0, // hostPort
    NULL, // userName
    NULL, // password,
    0, // options
    VIX_INVALID_HANDLE, // propertyListHandle
    NULL, // callbackProc
    NULL); // clientData

err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &hostHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
    "c:\Virtual Machines\vm1\win2000.vmx",
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &vmHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);
```
VixVM_PowerOff

Function Signature

VixHandle VixVM_PowerOff(VixHandle vmHandle,
                          VixVMPowerOpOptions powerOffOptions,
                          VixEventProc *callbackProc,
                          void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>powerOffOptions</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

- This function powers off a virtual machine. It is an asynchronous operation, and the job will be signalled when the operation completes.
- If you call this function while the virtual machine is powered off, the operation returns a VIX_E_VM_NOT_RUNNING error. You can safely ignore this error.
- If you call this function while the virtual machine is suspended, the operation returns a VIX_E_VM_NOT_RUNNING error. The virtual machine is not powered off and remains suspended.

Example

VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
                            VIX_SERVICEPROVIDER_VMWARE_SERVER,
                            NULL, // hostName
                            0, // hostPort
Chapter 4 Operations on Virtual Machines

NULL, // userName
NULL, // password,
0, // options
VIX_INVALID_HANDLE, // propertyListHandle
NULL, // callbackProc
NULL); // clientData

err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &hostHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
    "c:\Virtual Machines\vm1\win2000.vmx",
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &vmHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Assume this virtual machine is already running.
jobHandle = VixVM_PowerOff(vmHandle,
    0, // powerOffOptions,
    NULL, // callbackProc,
    NULL); // clientData
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
VixVM_PowerOn

Function Signature

VixHandle
VixVM_PowerOn(VixHandle vmHandle,
VixVMPowerOpOptions powerOnOptions,
VixHandle propertyListHandle,
VixEventProc *callbackProc,
void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>powerOnOptions</td>
<td>Must be VIX_VMPOWEROP_NORMAL.</td>
</tr>
<tr>
<td>propertyListHandle</td>
<td>Must be VIX_INVALID_HANDLE.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

- This function powers on a virtual machine. It is an asynchronous operation, and the job will be signalled when the operation completes.
- When you power on a virtual machine, the virtual machine is powered on independent of a console window. If a console window is open, it remains open. Otherwise, the virtual machine is powered on without a console window.
- This function can also be used to resume execution of a suspended virtual machine.

Example

VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
VIX_SERVICEPROVIDER_VMWARE_SERVER,
Chapter 4 Operations on Virtual Machines

VIX_INVALID_HANDLE, // propertyListHandle
0, // options
NULL, // userName
NULL, // password,
0, // hostPort
NULL, // hostName

err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &hostHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(handle, "c:\\Virtual Machines\\vm1\\win2000.vmx",
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &vmHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_PowerOn(vmHandle,
    VIX_VMPOWEROP_NORMAL, // powerOnOptions,
    VIX_INVALID_HANDLE, // propertyListHandle,
    NULL, // callbackProc,
    NULL); // clientData
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
VixVM_RemoveSnapshot

Function Signature

VixHandle
VixVM_RemoveSnapshot(VixHandle vmHandle,
VixHandle snapshotHandle,
int options,
VixEventProc *callbackProc,
void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>snapshotHandle</td>
<td>A handle to a snapshot. Call VixVM_GetRootSnapshot() to get a snapshot handle.</td>
</tr>
<tr>
<td>options</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

- This function deletes all saved states for the specified snapshot. If the snapshot was based on another snapshot, the base snapshot becomes the new root snapshot.
- A snapshot can be removed only while the associated virtual machine is powered off or suspended.
- This release of the Programming API can manage only a single snapshot for each virtual machine. A virtual machine imported from another VMware product can have more than one snapshot at the time it is imported. In that case, you can delete only a snapshot subsequently added using the Programming API.

Example

This example deletes every root snapshot belonging to a virtual machine.

VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
VixHandle snapshotHandle = VIX_INVALID_HANDLE;
int numRootSnapshots;
int index;

jobHandle = VixHost_Connect(VIX_API_VERSION,
    VIX_SERVICEPROVIDER_VMWARE_SERVER,
    NULL, // hostName
    0, // hostPort
    NULL, // userName
    NULL, // password,
    0, // options
    VIX_INVALID_HANDLE, // propertyListHandle
    NULL, // callbackProc
    NULL); // clientData

err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &hostHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
    "c:\Virtual Machines\vm1\win2000.vmx",
    NULL, // callbackProc
    NULL); // clientData
err = VixJob_Wait(jobHandle,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &vmHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Only 1 snapshot supported in this release.
numRootSnapshots = 1;
err = Vix_VM_GetNumRootSnapshots(vmHandle, &numRootSnapshots);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
for (index = 0; index < numRootSnapshots; index++) {

err = VixVM_GetRootSnapshot(vmHandle,
       index,
       &snapshotHandle);
if (VIX_OK != err) {
   // Handle the error...
   goto abort;
}
jobHandle = VixVM_RemoveSnapshot(vmHandle,
       snapshotHandle,
       0, // options
       NULL, // callbackProc
       NULL); // clientData
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
   // Handle the error...
   goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);
// Release the snapshot handle because we no longer need it.
Vix_ReleaseHandle(snapshotHandle);
snapshotHandle = VIX_INVALID_HANDLE;
}

abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(snapshotHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
VixVM_Reset

Function Signature

VixHandle VixVM_Reset(VixHandle vmHandle,
        VixVMPowerOpOptions powerOnOptions,
        VixEventProc *callbackProc,
        void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>powerOnOptions</td>
<td>Must be VIX_VMPOWEROP_NORMAL.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

This function resets a virtual machine, which is the equivalent of pressing the reset button on a physical machine. The reset is an asynchronous operation, and the job will be signalled when the operation completes. If the virtual machine is not powered on when you call this function, it returns an error.
VixVM_RevertToSnapshot

Function Signature

VixHandle VixVM_RevertToSnapshot(VixHandle vmHandle,
                                 VixHandle snapshotHandle,
                                 int options,
                                 VixHandle propertyListHandle,
                                 VixEventProc *callbackProc,
                                 void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>snapshotHandle</td>
<td>A handle to a snapshot. Call VixVM_GetRootSnapshot() to get a snapshot handle.</td>
</tr>
<tr>
<td>options</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>propertyListHandle</td>
<td>Must be VIX_INVALID_HANDLE.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

Restores the virtual machine to the state when the specified snapshot was created. This function can power on, power off, or suspend a virtual machine. The resulting power state reflects the power state when the snapshot was created.

Example

This example restores the virtual machine to the state of the first root snapshot.

VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
VixHandle snapshotHandle = VIX_INVALID_HANDLE;
int snapshotIndex;
jobHandle = VixHost_Connect(VIX_API_VERSION,
   VIX_SERVICEPROVIDER_VMWARE_SERVER,
   NULL, // hostName
   0, // hostPort
   NULL, // userName
   NULL, // password,
   0, // options
   VIX_INVALID_HANDLE, // propertyListHandle
   NULL); // clientData

err = VixJob_Wait(jobHandle,
   VIX_PROPERTY_JOB_RESULT_HANDLE,
   &hostHandle,
   VIX_PROPERTY_NONE);
if (VIX_OK != err) {
   // Handle the error...
   goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
   "c:\\Virtual Machines\\vm1\\win2000.vmx",
   NULL, // callbackProc
   NULL); // clientData
err = VixJob_Wait(jobHandle,
   VIX_PROPERTY_JOB_RESULT_HANDLE,
   &vmHandle,
   VIX_PROPERTY_NONE);
if (VIX_OK != err) {
   // Handle the error...
   goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Revert to snapshot #0.
snapshotIndex = 0;
err = VixVM_GetRootSnapshot(vmHandle,
   snapshotIndex,
   &snapshotHandle);
if (VIX_OK != err) {
   // Handle the error...
   goto abort;
}
jobHandle = VixVM_RevertToSnapshot(vmHandle,
   snapshotHandle,
   0, // options
   VIX_INVALID_HANDLE, // propertyListHandle
   &vmHandle,
   VIX_PROPERTY_NONE);
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(snapshotHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
VixVM_RunProgramInGuest

Function Signature

VixHandle VixVM_RunProgramInGuest(VixHandle vmHandle,
const char *guestProgramName,
const char *commandLineArgs,
VixRunProgramOptions options,
VixHandle propertyListHandle,
VixEventProc *callbackProc,
void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>guestProgramName</td>
<td>The path name of an executable file on the guest operating system.</td>
</tr>
<tr>
<td>commandLineArgs</td>
<td>A string to be passed as command line arguments to the executable identified by guestProgramName.</td>
</tr>
<tr>
<td>options</td>
<td>Run options for the program. See the notes below.</td>
</tr>
<tr>
<td>propertyList</td>
<td>Must be VIX_INVALID_HANDLE.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

- This function runs a program in the guest operating system. The program must be stored on a file system available to the guest before calling this function.
- You must call VixVM_LoginInGuest() before calling this function.
- If the options parameter is 0, this function will report completion to the job handle when the program exits in the guest operating system. Alternatively, you can pass VIX_RUNPROGRAM_RETURN_IMMEDIATELY as the value of the options parameter, and this function reports completion to the job handle as soon as the program starts in the guest.
Example

VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
                          VIX_SERVICEPROVIDER_VMWARE_SERVER,
                          NULL, // hostName
                          0, // hostPort
                          NULL, // userName
                          NULL, // password,
                          0, // options
                          VIX_INVALID_HANDLE, // propertyListHandle
                          NULL, // callbackProc
                          NULL); // clientData

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_JOB_RESULT_HANDLE,
                  &hostHandle,
                  VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
                       "c:\Virtual Machines\vm1\win2000.vmx",
                       NULL, // callbackProc
                       NULL); // clientData

err = VixJob_Wait(jobHandle,
                  VIX_PROPERTY_JOB_RESULT_HANDLE,
                  &vmHandle,
                  VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_PowerOn(vmHandle,
                          0, // powerOnOptions,
                          VIX_INVALID_HANDLE, // propertyListHandle,
                          NULL, // callbackProc,
                          NULL); // clientData

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {

Chapter 4 Operations on Virtual Machines

// Handle the error...
goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Wait until guest is completely booted.
jobHandle = VixVM_WaitForToolsInGuest(vmHandle,
    300, // timeoutInSeconds
    NULL, // callbackProc
    NULL); // clientData

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Authenticate for guest operations.
jobHandle = VixVM_LoginInGuest(vmHandle,
    "vixuser", // userName
    "secret", // password
    0, // options
    NULL, // callbackProc
    NULL); // clientData

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

// Run the target program.
jobHandle = VixVM_RunProgramInGuest(vmHandle,
    "c:\myProgram.exe",
    "/flag arg1 arg2",
    0, // options,
    VIX_INVALID_HANDLE, // propertyListHandle,
    NULL, // callbackProc,
    NULL); // clientData

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);
abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
**VixVM_Suspend**

**Function Signature**

```c
VixHandle VixVM_Suspend(VixHandle vmHandle,
                        VixVMPowerOpOptions powerOffOptions,
                        VixEventProc *callbackProc,
                        void *clientData);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>powerOffOptions</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

**Results**

VixHandle. A job handle that describes the state of this asynchronous operation.

**Notes**

This function suspends a virtual machine. It is an asynchronous operation, and the job will be signalled when the operation completes. If the virtual machine is not powered on when you call this function, the function returns an error.
VixVM_UpgradeVirtualHardware

Function Signature

VixHandle
VixVM_UpgradeVirtualHardware(VixHandle vmHandle,
int options,
VixEventProc *callbackProc,
void *clientData);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>options</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

Results

VixHandle. A job handle that describes the state of this asynchronous operation.

Notes

- Upgrades the virtual hardware version of the virtual machine to match the version of the Vix library. This has no effect if the virtual machine is already at the same version or at a newer version than the Vix library.
- For more information on the virtual hardware version, refer to the VMware Server Administration Guide.
- The virtual machine must be powered off to do this operation.

Example

VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
VIX_SERVICEPROVIDER_VMWARE_SERVER,
NULL, // hostName
0, // hostPort
NULL, // userName

VixVM_UpgradeVirtualHardware(vmHandle,
0,
NULL,
NULL);
VMware, Inc. 69

Chapter 4 Operations on Virtual Machines

NULL, // password,
0, // options
VIX_INVALID_HANDLE, // propertyListHandle
NULL, // callbackProc
NULL); // clientData

err = VixJob_Wait(jobHandle,
VIX_PROPERTY_JOB_RESULT_HANDLE,
&hostHandle,
VIX_PROPERTY_NONE);

if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
"c:\\Virtual Machines\\vm1\\win2000.vmx",
NULL, // callbackProc
NULL); // clientData

err = VixJob_Wait(jobHandle,
VIX_PROPERTY_JOB_RESULT_HANDLE,
&vmHandle,
VIX_PROPERTY_NONE);

if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_UpgradeVirtualHardware(vmHandle,
0, // options
NULL, // callbackProc
NULL); // clientData

err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);

if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
**VixVM_WaitForToolsInGuest**

**Function Signature**

```c
VixHandle VixVM_WaitForToolsInGuest(VixHandle vmHandle,
                                    int timeoutInSeconds,
                                    VixEventProc *callbackProc,
                                    void *clientData);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmHandle</td>
<td>Identifies a virtual machine. Call VixVM_Open() to create a virtual machine handle.</td>
</tr>
<tr>
<td>timeoutInSeconds</td>
<td>The timeout in seconds. If VMware Tools has not started by this time, the function completes with an error.</td>
</tr>
<tr>
<td>callbackProc</td>
<td>A callback function that will be invoked when the power operation is complete.</td>
</tr>
<tr>
<td>clientData</td>
<td>A parameter that will be passed to the callbackProc function.</td>
</tr>
</tbody>
</table>

**Results**

VixHandle. A job handle that describes the state of this asynchronous operation.

**Notes**

- This function signals the job when VMware Tools has successfully started in the guest operating system. VMware Tools is a collection of services that run in the guest.
- VMware Tools must be installed and running for some Vix functions to operate correctly. If VMware Tools is not installed in the guest operating system, or if the virtual machine is not powered on, this function reports an error to the job object.
- The VIX_PROPERTY_VM_TOOLS_STATE property of the virtual machine handle is undefined until VixVM_WaitForToolsInGuest() reports that VMware Tools is running.

**Example**

```c
VixError err = VIX_OK;
VixHandle jobHandle = VIX_INVALID_HANDLE;
int toolsState = 0;

// Resume suspended virtual machine.
```
jobHandle = VixVM_PowerOn(vmHandle,  
0, // powerOnOptions,  
VIX_INVALID_HANDLE, // propertyListHandle,  
NULL, // callbackProc,  
NULL); // clientData  
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);  
if (VIX_OK != err) {  
   // Handle the error...  
goto abort;  
}  

// Release the job handle because we no longer need it.  
Vix_ReleaseHandle(jobHandle);  

// Check if VMware Tools active.  
jobHandle = VixGetProperties(vmHandle,  
VIX_PROPERTY_VM_TOOLS_STATE,  
toolsState,  
VIX_PROPERTY_NONE);  
err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);  
if (VIX_OK != err) {  
   // Handle the error...  
goto abort;  
}  

// Release the job handle because we no longer need it.  
Vix_ReleaseHandle(jobHandle);  

if (VIX_TOOLSSTATE_RUNNING != toolsState) {  
   // Wait for VMware Tools to be active.  
   jobHandle = VixVM_WaitForToolsInGuest(VixHandle vmHandle,  
   120, // timeoutInSeconds,  
   NULL, // callbackProc,  
   NULL); // clientData  
   err = VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);  
   if (VIX_OK != err) {  
      // Handle the timeout...  
   }  
   // Release the job handle because we no longer need it.  
   Vix_ReleaseHandle(jobHandle);  
}
This chapter describes the API specifications for Programming API functions that are related to job handles. Job handles are returned by certain asynchronous Vix operations. The functions covered in this chapter are:

- “VixJob_CheckCompletion” on page 74
- “VixJob_GetError” on page 76
- “VixJob_Wait” on page 78
VixJob_CheckCompletion

Function Signature

VixError
VixJob_CheckCompletion(VixHandle jobHandle,
Bool *complete);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobHandle</td>
<td>The handle of a job object, returned from any asynchronous Vix function.</td>
</tr>
<tr>
<td>complete</td>
<td>An indicator of whether the job has completed.</td>
</tr>
</tbody>
</table>

Results

VixError. This function returns Vix_OK if it succeeded. Otherwise, it returns an appropriate error described in “Error Codes” on page 85.

Notes

This function performs a non-blocking test for completion of an asynchronous operation. It can be used to poll for completion in situations where a blocking function or a callback function is not desirable.

Example

```c
VixError err = Vix_OK;
VixError err1 = Vix_OK;
VixError err2 = Vix_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
Bool jobCompleted;

jobHandle = VixHost_Connect(VIX_API_VERSION,
VIX_SERVICEPROVIDER_VMWARE_SERVER,
NULL, // hostName
0, // hostPort
NULL, // userName
NULL, // password,
0, // options
VIX_INVALID_HANDLE, // propertyListHandle
NULL, // callbackProc
NULL); // clientData

err = VixJob_Wait(jobHandle,
VIX_PROPERTY_JOB_RESULT_HANDLE,
&jobCompleted);
```
&hostHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
    "c:\Virtual Machines\vm1\win2000.vmx",
    NULL, // callbackProc
    NULL); // clientData

jobCompleted = FALSE;
while (!jobCompleted) {
    err = VixJob_CheckCompletion(jobHandle, &jobCompleted);
    if (VIX_OK != err) {
        break;
    }
    // ...Do some processing, sleep, or pump user events...
}

// Get job results. (Same results as VixJob_Wait)
err = Vix_GetProperties(jobHandle,
    VIX_PROPERTY_JOB_RESULT_ERROR_CODE,
    &err1,
    VIX_PROPERTY_JOB_RESULT_HANDLE,
    &vmHandle,
    VIX_PROPERTY_NONE);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}

// Double-check error results from different functions.
err2 = VixJob_GetError(jobHandle);
ASSERT(err1 == err2);

abort:
    Vix_ReleaseHandle(jobHandle);
    Vix_ReleaseHandle(vmHandle);
    VixHost_Disconnect(hostHandle);
VixJob_GetError

Function Signature

VixError
VixJob_GetError(VixHandle jobHandle);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobHandle</td>
<td>The handle of a job object, returned from any asynchronous Vix function.</td>
</tr>
</tbody>
</table>

Results

VixError. The result returned by a completed asynchronous function.

Notes

- Applications should call this function only after an asynchronous job has completed. See also “VixJob_Wait” on page 78 or “VixJob_CheckCompletion” on page 74.
- The error code returned by this function is the same as the error code returned by VixJob_Wait().

Example

```c
VixError err = Vix_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;

jobHandle = VixHost_Connect(VIX_API_VERSION,
                           VIX_SERVICEPROVIDER_VMWARE_SERVER,
                           NULL, // hostName
                           0, // hostPort
                           NULL, // userName
                           NULL, // password,
                           0, // options
                           VIX_INVALID_HANDLE, // propertyListHandle
                           NULL, // callbackProc
                           NULL); // clientData

VixJob_Wait(jobHandle, VIX_PROPERTY_NONE);
err = VixJob_GetError(jobHandle);
if (VIX_OK != err) {
    // Handle the error...
    goto abort;
}
```
abort:
Vix_ReleaseHandle(jobHandle);
VixHost_Disconnect(hostHandle);
**VixJob_Wait**

**Function Signature**

```c
VixError VixJob_Wait(VixHandle jobHandle,
                      VixPropertyID firstPropertyID,
                      ...);
```

**Parameters**

<table>
<thead>
<tr>
<th>jobHandle</th>
<th>The handle of a job object, returned from any asynchronous Vix function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstPropertyID</td>
<td>The identity of a property, or else VIX_INVALID_PROPERTY.</td>
</tr>
</tbody>
</table>

**Results**

`VixError`. The error resulting from the asynchronous operation that returned the job handle.

**Notes**

- This function blocks until the job completes.
- If one or more property IDs are passed, their values are returned by this function. A property is requested by passing its ID in one parameter, followed by a reference to a storage location of the same type as the property value. After the last reference, terminate the list with the value `VIX_PROPERTY_NONE`.
- The error code returned by this function is the same as the error code returned by `VixJob_GetError`.
- The error code returned by this function is also available in the `Vix_JOB_RESULT_ERROR_CODE` property of the job object.

**Example**

```c
VixError err = VIX_OK;
VixHandle hostHandle = VIX_INVALID_HANDLE;
VixHandle jobHandle = VIX_INVALID_HANDLE;
VixHandle vmHandle = VIX_INVALID_HANDLE;
int errorLineNum;

jobHandle = VixHost_Connect(VIX_API_VERSION,
                           VIX_SERVICEPROVIDER_VMWARE_SERVER,
                           NULL, // hostName
                           0, // hostPort
                           NULL, // userName
```

```
err = VixJob_Wait(jobHandle,
   VIX_PROPERTY_JOB_RESULT_HANDLE,
   &hostHandle,
   VIX_PROPERTY_NONE);
if (VIX_OK != err) {
   // Handle the error...
   goto abort;
}
// Release the job handle because we no longer need it.
Vix_ReleaseHandle(jobHandle);

jobHandle = VixVM_Open(hostHandle,
   "c:\\Virtual Machines\\vm1\\win2000.vmx",
   NULL, // callbackProc
   NULL); // clientData
err = VixJob_Wait(jobHandle,
   VIX_PROPERTY_JOB_RESULT_HANDLE,
   &vmHandle,
   VIX_PROPERTY_NONE);
if (VIX_OK != err) {
   // Handle the error...
   goto abort;
}

abort:
Vix_ReleaseHandle(jobHandle);
Vix_ReleaseHandle(vmHandle);
VixHost_Disconnect(hostHandle);
APPENDIX A  Handle Types

Vix supports the following types of handles:

- **VIX_HOST_HANDLE**
  A single host machine that is running the Programming API. Currently, the Programming API supports only the local host machine, but this will be extended to describe a remote host or the host that is running a guest.

- **VIX_JOB_HANDLE**
  A job acts as both a semaphore and a result collector. A job is used to determine when an asynchronous operation has completed, and it saves the results of an asynchronous operation so a client can later retrieve them.

- **VIX_SNAPSHOT_HANDLE**
  A snapshot for a single virtual machine.

- **VIX_VM_HANDLE**
  A single virtual machine. The handle is valid whether the virtual machine is powered on or off.
This appendix lists the properties attached to the different kinds of handles used by the Programming API.

### Virtual Machine Properties
These properties are supported by handles of type `VIX_VM_HANDLE`.

<table>
<thead>
<tr>
<th>Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIX_PROPERTY_VM_NUM_VCPUS</td>
</tr>
<tr>
<td>VIX_PROPERTY_VM_VMX_PATHNAME</td>
</tr>
<tr>
<td>VIX_PROPERTY_VM_MEMORY_SIZE</td>
</tr>
<tr>
<td>VIX_PROPERTY_VM_POWER_STATE</td>
</tr>
<tr>
<td>VIX_PROPERTY_VM_TOOLS_STATE</td>
</tr>
<tr>
<td>VIX_PROPERTY_VM_IS_RUNNING</td>
</tr>
</tbody>
</table>

### Event Info Property
This property is returned in a property list referenced by the `moreEventInfo` parameter of the callback function signature. For more information on `moreEventInfo`, refer to “VixHost_FindItems” on page 20.

<table>
<thead>
<tr>
<th>Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIX_PROPERTY_FOUND_ITEM_LOCATION</td>
</tr>
</tbody>
</table>

### Job Properties
These properties are supported by handles of type `VIX_JOB_HANDLE`.

<table>
<thead>
<tr>
<th>Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIX_PROPERTY_JOB_RESULT_ERROR_CODE</td>
</tr>
<tr>
<td>VIX_PROPERTY_JOB_RESULT_VM_IN_GROUP</td>
</tr>
<tr>
<td>VIX_PROPERTY_JOB_RESULT_USER_MESSAGE</td>
</tr>
<tr>
<td>VIX_PROPERTY_JOB_RESULT_LINE_NUM</td>
</tr>
<tr>
<td>VIX_PROPERTY_JOB_RESULT_EXIT_CODE</td>
</tr>
<tr>
<td>VIX_PROPERTY_JOB_RESULT_COMMAND_OUTPUT</td>
</tr>
<tr>
<td>VIX_PROPERTY_JOB_RESULT_HANDLE</td>
</tr>
</tbody>
</table>
Snapshot Properties
These properties are supported by handles of type VIX_SNAPSHOT_HANDLE

<table>
<thead>
<tr>
<th>Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIX_SNAPSHOT_DISPLAYNAME_PROPERTY</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_DESCRIPTION_PROPERTY</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_POWERSTATE_PROPERTY</td>
</tr>
</tbody>
</table>
### Error Codes

<table>
<thead>
<tr>
<th>Error</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIX_E_ANON_GUEST_OPERATIONS_PROHIBITED_ON_HOST</td>
<td>Guest operations are not allowed for the anonymous user on any virtual machine on this host.</td>
</tr>
<tr>
<td>VIX_E_ANON_GUEST_OPERATIONS_PROHIBITED_ON_VM</td>
<td>Guest operations are not allowed for the anonymous user on this virtual machine.</td>
</tr>
<tr>
<td>VIX_E_BAD_VM_INDEX</td>
<td>The index parameter does not correspond to a result set.</td>
</tr>
<tr>
<td>VIX_E_CANCELLED</td>
<td>The operation was canceled.</td>
</tr>
<tr>
<td>VIX_E_CANNOT_CONNECT_TO_VM</td>
<td>Cannot connect to the virtual machine.</td>
</tr>
<tr>
<td>VIX_E_CANNOT_READ_VM_CONFIG</td>
<td>Cannot read the virtual machine configuration file.</td>
</tr>
<tr>
<td>VIX_E_DISK_FULL</td>
<td>An error occurred while writing a file because the disk is full. Data has not been saved. Free some space and try again.</td>
</tr>
<tr>
<td>VIX_E_FAIL</td>
<td>Internal error.</td>
</tr>
<tr>
<td>VIX_E_FILE_ACCESS_ERROR</td>
<td>You do not have access rights to the file.</td>
</tr>
<tr>
<td>VIX_E_FILE_ALREADY_EXISTS</td>
<td>The file already exists.</td>
</tr>
<tr>
<td>VIX_E_FILE_ALREADY_LOCKED</td>
<td>The file is already in use.</td>
</tr>
</tbody>
</table>
Table C-1. (Continued)

<table>
<thead>
<tr>
<th>Error</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIX_E_FILE_ERROR</td>
<td>A file access error occurred on the host or guest operating system.</td>
</tr>
<tr>
<td>VIX_E_FILE_NOT_FOUND</td>
<td>A file was not found.</td>
</tr>
<tr>
<td>VIX_E_FILE_READ_ONLY</td>
<td>The file is write-protected.</td>
</tr>
<tr>
<td>VIX_E_GUEST_OPERATIONS_PROHIBITED_ON_VM</td>
<td>Guest operations are not allowed on this virtual machine.</td>
</tr>
<tr>
<td>VIX_E_GUEST_OPERATIONS_PROHIBITED_ON_HOST</td>
<td>Guest operations are not allowed on any virtual machine on this host.</td>
</tr>
<tr>
<td>VIX_E_GUEST_USER_PERMISSIONS</td>
<td>Insufficient permissions in guest operating system.</td>
</tr>
<tr>
<td>VIX_E_HOST_USER_PERMISSIONS</td>
<td>Insufficient permissions in host operating system.</td>
</tr>
<tr>
<td>VIX_E_INCORRECT_FILE_TYPE</td>
<td>An error occurred while accessing a file. Wrong file type.</td>
</tr>
<tr>
<td>VIX_E_INVALID_ARG</td>
<td>One of the parameters is invalid.</td>
</tr>
<tr>
<td>VIX_E_INVALID_HANDLE</td>
<td>The handle is not a valid Vix object.</td>
</tr>
<tr>
<td>VIX_E_INVALID_PROPERTY_VALUE</td>
<td>Invalid property value.</td>
</tr>
<tr>
<td>VIX_E_INVALID_XML</td>
<td>Invalid file. Contents might be corrupt.</td>
</tr>
<tr>
<td>VIX_E_MISSING_ANON_GUEST_ACCOUNT</td>
<td>VM configuration must specify guest account name to be used for anonymous guest operations.</td>
</tr>
<tr>
<td>VIX_E_MISSING_REQUIRED_PROPERTY</td>
<td>This handle is missing a required property.</td>
</tr>
<tr>
<td>VIX_E_NO_GUEST_OS_INSTALLED</td>
<td>No operating system is installed in the virtual machine.</td>
</tr>
</tbody>
</table>
### Table C-1. Error Codes (Continued)

<table>
<thead>
<tr>
<th>Error</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIX_E_NOT_FOUND</td>
<td>Invalid file. A required section of the file is missing.</td>
</tr>
<tr>
<td>VIX_E_NOT_SUPPORTED</td>
<td>The operation is not supported for the specified parameters.</td>
</tr>
<tr>
<td>VIX_E_NOT_SUPPORTED_FOR_VM_VERSION</td>
<td>The operation is not supported for this virtual machine version.</td>
</tr>
<tr>
<td>VIX_E_NOT_SUPPORTED_ON_HANDLE_TYPE</td>
<td>The operation is not supported on this type of handle.</td>
</tr>
<tr>
<td>VIX_E_OBJECT_IS_BUSY</td>
<td>This function cannot be performed because the handle is in use by another function.</td>
</tr>
<tr>
<td>VIX_E_OP_NOT_SUPPORTED_ON_GUEST</td>
<td>The requested operation is not supported on this guest operating system.</td>
</tr>
<tr>
<td>VIX_E_OUT_OF_MEMORY</td>
<td>Memory allocation failed. Out of memory.</td>
</tr>
<tr>
<td>VIX_E_POWEROP_SCRIPTS_NOT_AVAILABLE</td>
<td>Cannot execute scripts.</td>
</tr>
<tr>
<td>VIX_E_PROGRAM_NOT_STARTED</td>
<td>A program could not run on the guest operating system.</td>
</tr>
<tr>
<td>VIX_E_READ_ONLY_PROPERTY</td>
<td>Cannot change a read-only property.</td>
</tr>
<tr>
<td>VIX_E_ROOT_GUEST_OPERATIONS_PROHIBITED_ON_HOST</td>
<td>Guest operations are not allowed for administrative user on any virtual machine on this host.</td>
</tr>
<tr>
<td>VIX_E_ROOT_GUEST_OPERATIONS_PROHIBITED_ON_VM</td>
<td>Guest operations are not allowed for administrative user on this virtual machine.</td>
</tr>
<tr>
<td>VIX_E_TEMPLATE_VM</td>
<td>Cannot perform this operation on a template virtual machine.</td>
</tr>
<tr>
<td>Error</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>VIX_E_TIMEOUT_WAITING_FORTOOLS</td>
<td>Timeout error while waiting for VMware Tools to start in the guest operating system.</td>
</tr>
<tr>
<td>VIX_E_TOO_MANY_HANDLES</td>
<td>Too many handles are open.</td>
</tr>
<tr>
<td>VIX_ETOOLS_NOT_RUNNING</td>
<td>VMware Tools is not running in the guest operating system.</td>
</tr>
<tr>
<td>VIXETYPE_MISMATCH</td>
<td>Invalid file. An object has the wrong type.</td>
</tr>
<tr>
<td>VIX_E_UNRECOGNIZED_COMMAND</td>
<td>An action contains an unrecognized command.</td>
</tr>
<tr>
<td>VIX_E_UNRECOGNIZED_COMMAND</td>
<td>An action contains an unrecognized command</td>
</tr>
<tr>
<td>VIX_E_UNRECOGNIZEDPROPERTY</td>
<td>Unrecognized handle property identifier.</td>
</tr>
<tr>
<td>VIX_E_VM_ALREADY_LOADED</td>
<td>The virtual machine has already been loaded.</td>
</tr>
<tr>
<td>VIX_E_VM_INSUFFICIENT_HOST_MEMORY</td>
<td>Not enough physical memory is available to power on this virtual machine.</td>
</tr>
<tr>
<td>VIX_E_VM_IS_RUNNING</td>
<td>The virtual machine should not be powered on. It appears to be running already.</td>
</tr>
<tr>
<td>VIX_E_VM_NOT_ENOUGH_CPUS</td>
<td>This virtual machine is configured to run with 2 CPUs, but the host only has 1 CPU. The virtual machine cannot be powered on.</td>
</tr>
<tr>
<td>VIX_E_VM_NOT_FOUND</td>
<td>The virtual machine cannot be found.</td>
</tr>
<tr>
<td>VIX_E_VM_NOT_RUNNING</td>
<td>The virtual machine needs to be powered on for this function.</td>
</tr>
<tr>
<td>Error</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>VIX_OK</td>
<td>The operation was successful.</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_CONFIG</td>
<td>Something is wrong with the configuration file</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_DISKLIB</td>
<td>Disk problem</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_DISKLOCKED</td>
<td>One or more of the disks are busy</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_DUMPER</td>
<td>Unable to open snapshot file</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_DUPLICATEDDISK</td>
<td>The virtual disk is used multiple times</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_EXISTS</td>
<td>A file by that name already exists</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_INCONSISTENT</td>
<td>The snapshot files are in an inconsistent state</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_INDEPENDENTDISK</td>
<td>Powered on snapshot are not allowed on virtual machines with independent disks</td>
</tr>
<tr>
<td>VIX_SNAPSHOTINVAL</td>
<td>Internal snapshot error</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_LOCKED</td>
<td>A snapshot operation is already in progress</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_NAMEETOOLONG</td>
<td>The filename is too long</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_NOCHANGE</td>
<td>The state of the virtual machine has not changed since the last snapshot operation</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_NONUNIQUE_NAME</td>
<td>The name does not uniquely identify one snapshot</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_NOPERM</td>
<td>Insufficient permissions</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_NOTFOUND</td>
<td>A file by that name does not exist</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_VERSION</td>
<td>Snapshots are not allowed on this virtual machine</td>
</tr>
<tr>
<td>VIX_SNAPSHOT_VIXFILE</td>
<td>Cannot snapshot all metadata files</td>
</tr>
</tbody>
</table>
Table C-1. (Continued)

<table>
<thead>
<tr>
<th>Error</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIX_VM_IS_IN_VMTEAM</td>
<td>The virtual machine belongs to a Team</td>
</tr>
<tr>
<td>VIX_VM_IS_NOT_IN_VMTEAM</td>
<td>The virtual machine does not belong to a Team</td>
</tr>
<tr>
<td>VIX_VMTEAM_IS_IN_POWER_OP</td>
<td>The team is in the middle of a power operation</td>
</tr>
<tr>
<td>VIX_VMTEAM_IS_RUNNING</td>
<td>The team is powered on</td>
</tr>
</tbody>
</table>
APPENDIX D  The Simple Perl API

About the Simple Perl API

The Simple Perl API allows you to write simple Vix applications in Perl. This API provides a subset of the functionality present in the Programming API.

The Simple Perl API is implemented as a wrapper to the Programming API. All functions in the Simple Perl API are blocking calls, with no callbacks and no completion calls.

Installing the Simple Perl API

The generic process for installing this module is the following:

$ tar -zxvf vix-perl.tar.gz
$ cd vix-perl
$ perl Makefile.PL
$ make
$ make install

The above procedure works for the Linux platform, assuming your Perl installation is correct. Most distributions have all tools installed and correctly configured by default.

For the Microsoft Windows platform, you need a Perl installation, a way to uncompress tar archives (most zip programs will do so), and a C build environment. Most likely, your build environment is an installation of Visual Studio. If you have another build environment (for example, cygwin, or independently downloaded build tools), your build process will be different.

Visual Studio installs a small script that starts a shell with all appropriate environment variables defined. If you are using Visual Studio, start a Visual Studio build shell and follow this procedure:

> (decompress vix-perl.tar)
> perl Makefile.PL
> nmake
> nmake install

Because this module is distributed as source, you can specify your own arguments at any of these steps, to use optimization or debugging flags, for example, or to override installation directories.
There are several other ways to build. See the Perl document perlmodinstall for more ideas at http://perldoc.perl.org/perlmodinstall.html.

Perl 5.8.0 and Perl 5.8.1
This package has been tested against Perl 5.8.1 on the Linux platform and Perl 5.8.0 on the Microsoft Windows platform. No advanced Perl features are present, so this package should work with older versions of Perl but has not been tested against any Perl version before Perl 5.8.0.

Simple Perl API Quick Reference Guide

Name and Description
VMware::Vix::Simple – Provides a simple interface to the Programming API. It hides the underlying asynchronous nature of the API while still providing the core functionality.

General Functions
VMware::Vix::Simple includes the following general functions (in alphabetical order).

FindRunningVMs
@vmlist = FindRunningVMs($hostHandle, $timeoutInSeconds);
Returns the URLs of the virtual machines running on the host associated with the $hostHandle. The first element in the list is an error code.

GetErrorText
$errstring = GetErrorText($err);
Returns the human-readable error string associated with an error code.

GetHandleType
$type = GetHandleType($handle);
Returns the type of handle referred to by $handle.

GetProperties
@properties = GetProperties($handle, prop1, prop2, ...);
Returns the values of the requested properties from the object pointed to by $handle. The first element in the list is an error code.
Appendix D The Simple Perl API

HostConnect

($err, $hostHandle) = HostConnect(VIX_API_VERSION, $hostType, $hostname, $hostport, $username, $password, $options, $propertyListHandle);

Initializes a remote host object. You must call this function before calling any other Vix API. The host object will be used for all Vix operations.

Returns an error code and the host handle. $hostType should be VIX_SERVICEPROVIDER_VMWARE_SERVER. $options should be 0. $propertyListHandle should be VIX_INVALID_HANDLE.

HostDisconnect

HostDisconnect($hostHandle);

Shuts down a host handle returned by HostConnect.

RegisterVM

$err = RegisterVM($hostHandle, $vmxFilePath);

Registers the virtual machine with the host associated with the hostHandle. Virtual machines must be registered before they can be opened.

ReleaseHandle

ReleaseHandle($handle);

Can be used to release any handle when it is no longer needed. Do not use a handle after it has been Released.

UnregisterVM

$err = UnregisterVM($hostHandle, $vmxFilePath);

Unregisters the virtual machine with the host associated with the hostHandle.

Virtual Machines Functions

VMware::Vix::Simple includes the following virtual machine functions (in alphabetical order).

VMCopyFileFromGuestToHost

$err = VMCopyFileFromGuestToHost($vmHandle, $guestFilePath, $hostFilePath, $options, $propertyListHandle);
This function copies $guestFilePath to $hostFilePath. If $options is VIX_RUNPROGRAM_RETURN_IMMEDIATELY, the function does not wait for the copy to complete before returning. $propertyListHandle should be VIX_INVALID_HANDLE. LoginInGuest() must be called first.

**VMCopyFileFromHostToGuest**

```php
$err = VMCopyFileFromHostToGuest($vmHandle, $hostFilePath, $guestFilePath, $options, $propertyListHandle);
```

This function copies $hostFilePath to $guestFilePath. If $options is VIX_RUNPROGRAM_RETURN_IMMEDIATELY, the function does not wait for the copy to complete before returning. $propertyListHandle should be VIX_INVALID_HANDLE. LoginInGuest() must be called first.

**VMDelete**

```php
$err = VMDelete($vmHandle, $options);
```

Deletes the virtual machine $vmHandle returned by the VMOpen function. $options should be 0.

**VMLoginInGuest**

```php
$err = VMLoginInGuest($vmHandle, $userName, $password, $options);
```

Validates a connection on the guest operating system for further guest OS operations. $options should be 0.

**VMOpen**

```php
($err, $vmHandle) = VMOpen($hostHandle, $vmxPathName);
```

Opens the virtual machine in $vmxPathName on the host identified by $hostHandle.

**VMPowerOff**

```php
$err = VMPowerOff($vmHandle, $options);
```

Powers off the virtual machine. $options should be 0.

**VMPowerOn**

```php
$err = VMPowerOn($vmHandle, $options, $propertyListHandle);
```

Powers on the virtual machine. $options should be 0. $propertyListHandle should be VIX_INVALID_HANDLE.
VMReset

```perl
$err = VMReset($vmHandle, $options);
```

Restarts the virtual machine. $options should be 0.

VMRunProgramInGuest

```perl
$err = VMRunProgramInGuest($vmHandle, $guestProgramName, $commandLineArgs, $options, $propertyListHandle);
```

This function runs $guestProgramName with $commandLineArgs. If $options is VIX_RUNPROGRAM_RETURN_IMMEDIATELY, the function does not wait for the $guestProgramName to complete before returning. $propertyListHandle should be VIX_INVALID_HANDLE. LoginInGuest() must be called first.

VMSuspend

```perl
$err = VMSuspend($vmHandle, $options);
```

Suspends the virtual machine. $options should be 0.

VMWaitForToolsInGuest

```perl
$err = VMWaitForToolsInGuest($vmHandle, $timeoutInSeconds);
```

This function will return when the VMware Tools has successfully started in the guest or when the $timeoutInSeconds have elapsed.

**Virtual Machine Snapshot Functions**

VMware::Vix::Simple includes the following virtual machine snapshot functions (in alphabetical order).

VMCreateSnapshot

```perl
($err, $snapshotHandle) = VMCreateSnapshot($vmHandle, $name, $description, $options, $propertyListHandle);
```

Creates a new snapshot of the current state of the virtual machine and stores the snapshot as $name and $description. $options should be 0. $propertyListHandle should be VIX_INVALID_HANDLE. Returns an error code and the handle to the newly created snapshot.

VMGetNumRootSnapshots

```perl
($err, $numRootSnapshots) = VMGetNumRootSnapshots($vmHandle);
```

Returns the number of top-level snapshots in a virtual machine.
VMGetRootSnapshot

($err, $snapshotHandle) = VMGetRootSnapshot($vmHandle, $index);

Returns a handle to the $index snapshot (numbered from 0 to n-1).

VMInstallTools

$err = VMInstallTools($vmHandle, $options, $commandLineArgs);

Installs the latest version of the tools onto the virtual machine. The virtual machine must be powered on. $options should be 0. $commandLineArgs should be undef.

VMRemoveSnapshot

$err = VMRemoveSnapshot($vmHandle, $snapshotHandle, $options);

Removes the snapshot from a virtual machine. $options should be 0.

VMRevertToSnapshot

$err = VMRevertToSnapshot($vmHandle, $snapshotHandle, $options, $propertyListHandle);

Reverts the state of the virtual machine to the state when the snapshot was created. $options should be 0. $propertyListHandle should be VIX_INVALID_HANDLE.

VMUpgradeVirtualHardware

$err = VMUpgradeVirtualHardware($vmHandle, $options);

Upgrade the virtual hardware of the virtual machine. The virtual machine must be powered off to perform this operation. $options should be 0.
APPENDIX E  The vmrun Utility

Command-Line Application: vmrun

VMware Vix includes a separate application, vmrun, for operating virtual machines from the command line.

To start the vmrun application

From the command prompt:

    vmrun [<authentication flags>] <command> [<option>]

The flags, commands, and options are described in the following tables.

Authentication Flags

Valid authentication flags are defined in the following table:

Table E-1.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Specifies a host to which vmrun should connect.</td>
<td>host name</td>
</tr>
<tr>
<td>-P</td>
<td>Specifies a port on the host to which vmrun should connect.</td>
<td>port number</td>
</tr>
<tr>
<td>-u</td>
<td>Specifies the user as which vmrun should authenticate.</td>
<td>user name</td>
</tr>
<tr>
<td>-p</td>
<td>Supplies the password to match the user name.</td>
<td>password</td>
</tr>
</tbody>
</table>

Commands and Options

Valid vmrun commands and corresponding options are described in the following table:

Table E-2.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Lists all running virtual machines.</td>
<td>None</td>
</tr>
<tr>
<td>start</td>
<td>Starts a virtual machine.</td>
<td>Path to vmx file</td>
</tr>
<tr>
<td>stop</td>
<td>Stops a virtual machine.</td>
<td>Path to vmx file</td>
</tr>
<tr>
<td>reset</td>
<td>Resets a virtual machine.</td>
<td>Path to vmx file</td>
</tr>
<tr>
<td>suspend</td>
<td>Suspends a virtual machine.</td>
<td>Path to vmx file</td>
</tr>
<tr>
<td>snapshot</td>
<td>Creates a snapshot of a virtual machine.</td>
<td>Path to vmx file</td>
</tr>
</tbody>
</table>
### Table E-2. (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteSnapshot</td>
<td>Removes a snapshot from a virtual machine.</td>
<td>Path to vmx file</td>
</tr>
<tr>
<td>revertToSnapshot</td>
<td>Sets virtual machine state to a snapshot.</td>
<td>Path to vmx file</td>
</tr>
<tr>
<td>installtools</td>
<td>Mounts VMware Tools image in a guest operating system.</td>
<td>Path to vmx file</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You must install VMware Tools from within the guest operating system.</td>
<td></td>
</tr>
<tr>
<td>upgradevm</td>
<td>Upgrades a virtual machine to the current product version.</td>
<td>Path to vmx file</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You must upgrade VMware Tools from within the guest operating system.</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE** Before running this command on a Microsoft Windows host, you must do one of the following:

- Change your working directory to the Programming API directory. For example:
  ```cmd
  c:\Program Files\Vmware\Vmware Vix
  ```

- Add the Programming API directory to the system path. On Windows 2000 and XP, this setting is changed in:
  ```cmd
  Control Panels > System > Advanced > Environment Variables > System variables > Path
  ```

**Examples for vmrun**

For example, type this command to start a virtual machine:

- In a Linux terminal:
  ```bash
  vmrun start /usr/local/VMs/<virtual_machine_name>.vmx
  ```

- On the Microsoft Windows command line:
  ```cmd
  vmrun start "c:\My Virtual Machines\<virtual_machine_name>.vmx"
  ```

**NOTE** The `vmrun` command might time out (fail) with virtual machines that require input through a dialog box. To disable dialog boxes, insert the following line into the .vmx configuration file for a virtual machine:

```python
msg.autoAnswer = TRUE
```
Tips

- When using a path name that contains spaces, use quotation marks to delineate the entire string.

- When using vmrun on a Microsoft Windows host, remember to use quotation marks or double backslash characters in path names that include a backslash character.
The following table lists the revision history for the Programming API Reference Guide.

**Table F-1.**

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>August 11, 2006</td>
<td>Removed 2 functions not available in the API. Added handle release comments to samples.</td>
</tr>
<tr>
<td>July 27, 2006</td>
<td>Corrected example for VixVM_Delete().</td>
</tr>
<tr>
<td>July 6, 2006</td>
<td>The first release of this document</td>
</tr>
</tbody>
</table>
Index

A
account name, validating 46
authentication 46

C
command-line application
  vmrun 97
compatibility with VMware products 2
connecting to a host 16

D
data types, identifying 11
deleting virtual machines 37
disconnecting from a host 18
document set 2

E
error checking 74, 76
error codes 85

F
file name format (copying) 26, 30
files, copying
  from guest to host 26
  from host to guest 30
finding items on a host 20
finding Vix objects 20

G
guest operations
  copying files from host 30
  copying files to host 26

H
logging in 46
running programs 63
hand types 81
handles
  getting handle type 8
  getting properties from 9
  handle types 81
  releasing 13
host connection 16, 18
hosts
  deleting virtual machines 37
  disconnect 18
  initializing 16
  opening a virtual machine 50
  registering virtual machines 23
  unregistering virtual machines 24

I
initializing the host object 16
installation 3
intended users 1

J
job completion 74, 78

L
language code 7

P
password, validating 46
path name format 50
Perl API 91
  described 91
  installation 91
  reference 92
power operations 52, 54
  power off 52
  power on 54
  reset 59
  suspend 67
Programming API
  description 1
  installation 3
properties
  data types, identifying 11
  event info 83
  job properties 83
  virtual machine 83
R
  registering virtual machines 23
  resetting virtual machines 59, 97
  resuming virtual machines 54
S
  snapshots
    creating 34
    removing 56
    reverting to 60
    root snapshots number, getting 39
    root, getting 40
  starting virtual machines 54, 97
  stopping virtual machines 52, 97
string buffers
  allocating 9
  freeing 6
  suspending virtual machines 67, 97
T
  troubleshooting 85
U
  unregistering virtual machines 24
  upgrading virtual hardware 68
  upgrading virtual machines 98
  users, intended 1
V
  virtual hardware, upgrading 68
  virtual machines
    deleting 37
    registering 23
    resetting 59
    resuming 54
    starting 54
    stopping 52
    suspending 67
    taking snapshots 34
    unregistering 24
    upgrading 98
Vix 1
  Vix objects, finding 20
  Vix_FreeBuffer 6
  Vix_GetErrorText 7
  Vix_GetHandleType 8
  Vix_GetProperties 9
  Vix_GetPropertyType 11
  Vix_PumpEvents 12
  Vix_ReleaseHandle 13
  VixHost_Connect 16
  VixHost_Disconnect 18
  VixHost_FindItems 20
  VixHost_RegisterVM 23
  VixHost_UnregisterVM 24
  VixJob_CheckCompletion 74
  VixJob_GetError 76
Index

VixJob_Wait 78
VixVM_CopyFileFromGuestToHost 26
VixVM_CopyFileFromHostToGuest 30
VixVM_CreateSnapshot 34
VixVM_Delete 37
VixVM_GetNumRootSnapshots 39
VixVM_GetRootSnapshot 40
VixVM_InstallTools 43
VixVM_LoginInGuest 46
VixVM_Open 50
VixVM_PowerOff 52
VixVM_PowerOn 54
VixVM_RemoveSnapshot 56
VixVM_Reset 59
VixVM_RevertToSnapshot 60
VixVM_RunProgramInGuest 63
VixVM_Suspend 67
VixVM_UpgradeVirtualHardware 68
VixVM_WaitForToolsInGuest 70
vmrun utility 97

VMware Tools
    and copying files 27, 30
    and logging in 46
    installing 43
    job, signalling 70