

Interconnect Library

Generated by Doxygen 1.13.2

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 BaseManagerWithConnection Class Reference	7
4.1.1 Detailed Description	8
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 BaseManagerWithConnection()	8
4.1.3 Member Function Documentation	8
4.1.3.1 updateConnection()	8
4.2 ConnectionInfo Struct Reference	8
4.2.1 Detailed Description	9
4.3 ConnectionManager Class Reference	9
4.3.1 Detailed Description	10
4.3.2 Constructor & Destructor Documentation	10
4.3.2.1 ConnectionManager()	10
4.3.3 Member Function Documentation	10
4.3.3.1 getConnection()	10
4.3.3.2 getConnectionInfo()	10
4.3.3.3 initializeConnection()	10
4.3.3.4 isConnectionAlive()	11
4.4 ConnectionManagerTests Class Reference	11
4.5 ExecutionInfo Struct Reference	11
4.5.1 Detailed Description	12
4.6 ExecutionInfoObtainer Class Reference	12
4.6.1 Detailed Description	12
4.6.2 Member Function Documentation	12
4.6.2.1 runAndObtainExecutionInfo()	12
4.7 ExecutionInfoObtainerTests Class Reference	13
4.8 ILibpcapWrapper Class Reference	13
4.8.1 Detailed Description	14
4.8.2 Member Function Documentation	14
4.8.2.1 close()	14
4.8.2.2 closeHandler()	14
4.8.2.3 getLinkLayerType()	14
4.8.2.4 listenForPackets()	14
4.8.2.5 openHandlerLive()	15

4.9 ILibvirtWrapper Class Reference	15
4.9.1 Detailed Description	17
4.9.2 Member Function Documentation	17
4.9.2.1 attachDeviceToVm()	17
4.9.2.2 connectionIsAlive()	17
4.9.2.3 connectOpen()	18
4.9.2.4 createNetworkFromXml()	18
4.9.2.5 createNewStream()	18
4.9.2.6 createVirtualMachineFromXml()	18
4.9.2.7 destroyNetwork()	19
4.9.2.8 detachDeviceFromVm()	19
4.9.2.9 domainGetInfo()	19
4.9.2.10 domainLookupByName()	20
4.9.2.11 domainLookupByUuid()	20
4.9.2.12 finishAndFreeStream()	20
4.9.2.13 freeDomain()	21
4.9.2.14 getConnectUrl()	21
4.9.2.15 getDomainName()	21
4.9.2.16 getDomainUUID()	21
4.9.2.17 getDriverType()	22
4.9.2.18 getDriverVersion()	22
4.9.2.19 getLastErrorMessage()	23
4.9.2.20 getLibVersion()	23
4.9.2.21 getListOfAllDomains()	23
4.9.2.22 getNetworkByName()	23
4.9.2.23 getNetworkDefinition()	24
4.9.2.24 getNodeInfo()	24
4.9.2.25 getUuidFromDomain()	24
4.9.2.26 openDomainConsole()	25
4.9.2.27 receiveDataFromStream()	25
4.9.2.28 sendDataToStream()	25
4.9.2.29 updateVmDevice()	26
4.10 LibpcapWrapper Class Reference	26
4.10.1 Detailed Description	27
4.10.2 Member Function Documentation	27
4.10.2.1 close()	27
4.10.2.2 closeHandler()	27
4.10.2.3 getLinkLayerType()	27
4.10.2.4 listenForPackets()	28
4.10.2.5 openHandlerLive()	28
4.11 LibvirtWrapper Class Reference	29
4.11.1 Detailed Description	30

4.11.2 Member Function Documentation	30
4.11.2.1 attachDeviceToVm()	30
4.11.2.2 connectionIsAlive()	31
4.11.2.3 connectOpen()	31
4.11.2.4 createNetworkFromXml()	31
4.11.2.5 createNewStream()	31
4.11.2.6 createVirtualMachineFromXml()	32
4.11.2.7 destroyNetwork()	32
4.11.2.8 detachDeviceFromVm()	32
4.11.2.9 domainGetInfo()	33
4.11.2.10 domainLookupByName()	33
4.11.2.11 domainLookupByUuid()	33
4.11.2.12 finishAndFreeStream()	34
4.11.2.13 freeDomain()	34
4.11.2.14 getConnectUrl()	34
4.11.2.15 getDomainName()	35
4.11.2.16 getDomainUUID()	36
4.11.2.17 getDriverType()	36
4.11.2.18 getDriverVersion()	36
4.11.2.19 getLastErr()	37
4.11.2.20 getLibVersion()	37
4.11.2.21 getListOfAllDomains()	37
4.11.2.22 getNetworkByName()	38
4.11.2.23 getNetworkDefinition()	38
4.11.2.24 getNodeInfo()	38
4.11.2.25 getUuidFromDomain()	39
4.11.2.26 openDomainConsole()	39
4.11.2.27 receiveDataFromStream()	39
4.11.2.28 sendDataToStream()	40
4.11.2.29 updateVmDevice()	40
4.12 LibvirtWrapperMock Class Reference	41
4.13 ListenCallbackArgs Struct Reference	43
4.13.1 Detailed Description	43
4.14 NetworkDefinition Struct Reference	43
4.14.1 Detailed Description	44
4.15 Packet Struct Reference	44
4.15.1 Detailed Description	44
4.16 PacketSniffer Class Reference	44
4.16.1 Detailed Description	45
4.16.2 Member Function Documentation	45
4.16.2.1 closeAndStopListening()	45
4.16.2.2 getNumberOfReceivedPackets()	45

4.16.2.3 getPacketFromQueue()	46
4.16.2.4 listenForPacket()	46
4.16.2.5 openSnifferHandler()	46
4.17 PacketSnifferException Class Reference	47
4.17.1 Detailed Description	47
4.17.2 Constructor & Destructor Documentation	47
4.17.2.1 PacketSnifferException()	47
4.17.3 Member Function Documentation	47
4.17.3.1 what()	47
4.18 StreamData Struct Reference	48
4.18.1 Detailed Description	48
4.19 StringUtils Class Reference	48
4.19.1 Detailed Description	48
4.19.2 Member Function Documentation	48
4.19.2.1 copyStringToCharArray()	48
4.20 StringUtilsTests Class Reference	49
4.21 TestingUtils Class Reference	49
4.22 Version Struct Reference	49
4.22.1 Detailed Description	50
4.23 VersionUtils Class Reference	50
4.23.1 Detailed Description	50
4.23.2 Member Function Documentation	50
4.23.2.1 getVersion()	50
4.24 VersionUtilsTests Class Reference	51
4.25 VirtualizationException Class Reference	51
4.25.1 Detailed Description	52
4.25.2 Constructor & Destructor Documentation	52
4.25.2.1 VirtualizationException()	52
4.25.3 Member Function Documentation	52
4.25.3.1 what()	52
4.26 VirtualizationFacade Class Reference	52
4.26.1 Detailed Description	53
4.26.2 Constructor & Destructor Documentation	54
4.26.2.1 VirtualizationFacade()	54
4.26.3 Member Function Documentation	55
4.26.3.1 attachDeviceToVm()	55
4.26.3.2 closeStream()	55
4.26.3.3 createVirtualMachine()	55
4.26.3.4 createVirtualNetworkFromXml()	55
4.26.3.5 destroyNetwork()	56
4.26.3.6 detachDeviceFromVm()	56
4.26.3.7 getConnectionInfo()	56

4.26.3.8 getInfoAboutVirtualMachine()	56
4.26.3.9 getListOfVirtualMachinesWithInfo()	57
4.26.3.10 getNetworkDefinition()	57
4.26.3.11 initializeConnection()	57
4.26.3.12 isConnectionAlive()	57
4.26.3.13 openVirtualMachineConsole()	58
4.26.3.14 receiveDataFromConsole()	58
4.26.3.15 sendDataToConsole()	58
4.26.3.16 updateVmDevice()	58
4.27 VirtualMachineConsoleManager Class Reference	59
4.27.1 Detailed Description	60
4.27.2 Member Function Documentation	60
4.27.2.1 BaseManagerWithConnection()	60
4.27.2.2 closeStream()	60
4.27.2.3 getDataFromStream()	60
4.27.2.4 openVirtualMachineConsole()	61
4.27.2.5 sendDataToStream()	61
4.28 VirtualMachineConsoleManagerTests Class Reference	61
4.29 VirtualMachineInfo Struct Reference	62
4.29.1 Detailed Description	62
4.30 VirtualMachineManager Class Reference	62
4.30.1 Detailed Description	63
4.30.2 Member Function Documentation	63
4.30.2.1 attachDeviceToVirtualMachine()	63
4.30.2.2 BaseManagerWithConnection()	64
4.30.2.3 createVirtualMachine()	64
4.30.2.4 detachDeviceFromVirtualMachine()	64
4.30.2.5 getInfoAboutVirtualMachine()	64
4.30.2.6 getListOfVirtualMachinesWithInfo()	65
4.30.2.7 updateVmDevice()	65
4.31 VirtualMachineManagerMockGetInfoAboutVirtualMachine Class Reference	65
4.32 VirtualMachineManagerTests Class Reference	67
4.33 VirtualNetworkManager Class Reference	67
4.33.1 Detailed Description	68
4.33.2 Member Function Documentation	68
4.33.2.1 BaseManagerWithConnection()	68
4.33.2.2 createNetworkFromXml()	68
4.33.2.3 destroyNetwork()	69
4.33.2.4 getNetworkXmlDefinition()	69
5 File Documentation	71
5.1 PacketSnifferException.h	71

5.2 VirtualizationException.h	71
5.3 ILibpcapWrapper.h	71
5.4 ILibvirtWrapper.h	72
5.5 ConnectionInfo.h	73
5.6 ExecutionInfo.h	73
5.7 ListenCallbackArgs.h	73
5.8 NetworkDefinition.h	73
5.9 Packet.h	74
5.10 StreamData.h	74
5.11 Version.h	74
5.12 VirtualMachineInfo.h	74
5.13 PacketSniffer.h	75
5.14 ExecutionInfoObtainer.h	75
5.15 StringUtils.h	75
5.16 VersionUtils.h	75
5.17 BaseManagerWithConnection.h	76
5.18 ConnectionManager.h	76
5.19 VirtualMachineConsoleManager.h	77
5.20 VirtualMachineManager.h	77
5.21 VirtualNetworkManager.h	77
5.22 VirtualizationFacade.h	78
5.23 LibpcapWrapper.h	79
5.24 LibvirtWrapper.h	79
5.25 LibvirtWrapperMock.h	80
5.26 VirtualMachineManagerMockGetInfoAboutVirtualMachine.h	80
5.27 TestingUtils.h	81
Index	83

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BaseManagerWithConnection	7
VirtualMachineConsoleManager	59
VirtualMachineManager	62
VirtualMachineManagerMockGetInfoAboutVirtualMachine	65
VirtualNetworkManager	67
ConnectionInfo	8
ConnectionManager	9
std::exception	
PacketSnifferException	47
VirtualizationException	51
ExecutionInfo	11
ExecutionInfoObtainer	12
ILibpcapWrapper	13
LibpcapWrapper	26
ILibvirtWrapper	15
LibvirtWrapper	29
LibvirtWrapperMock	41
ListenCallbackArgs	43
NetworkDefinition	43
Packet	44
PacketSniffer	44
StreamData	48
StringUtils	48
testing::Test	
ConnectionManagerTests	11
ExecutionInfoObtainerTests	13
StringUtilsTests	49
VersionUtilsTests	51
VirtualMachineConsoleManagerTests	61
VirtualMachineManagerTests	67
VirtualMachineManagerTests	67
TestingUtils	49
Version	49
VersionUtils	50
VirtualizationFacade	52
VirtualMachineInfo	62

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

BaseManagerWithConnection	Base class for managers that require a libvirt connection	7
ConnectionInfo	Structure containing information about a hypervisor connection	8
ConnectionManager	Manages connections to libvirt hypervisors	9
ConnectionManagerTests	11
ExecutionInfo	Structure containing execution result information	11
ExecutionInfoObtainer	Utility class for executing functions and capturing execution information	12
ExecutionInfoObtainerTests	13
ILibpcapWrapper	Interface for libpcap wrapper functionality	13
ILibvirtWrapper	Interface for Libvirt wrapper functionality	15
LibpcapWrapper	Concrete implementation of the ILibpcapWrapper interface	26
LibvirtWrapper	Concrete implementation of the ILibvirtWrapper interface	29
LibvirtWrapperMock	41
ListenCallbackArgs	Structure containing arguments for packet listening callbacks	43
NetworkDefinition	Structure containing network definition in XML format	43
Packet	Structure representing a captured network packet	44
PacketSniffer	Class for capturing network packets	44
PacketSnifferException	Exception class for packet sniffer-related errors	47
StreamData	Structure containing data received from a virtual machine console stream	48
StringUtils	Utility class for string manipulation operations	48

StringUtilsTests	49
TestingUtils	49
Version	
Structure representing a semantic version number	49
VersionUtils	
Utility class for version number operations	50
VersionUtilsTests	51
VirtualizationException	
Exception class for virtualization-related errors	51
VirtualizationFacade	
Facade class providing a unified interface for virtualization operations	52
VirtualMachineConsoleManager	
Manages virtual machine console connections	59
VirtualMachineConsoleManagerTests	61
VirtualMachineInfo	
Structure containing basic information about a virtual machine	62
VirtualMachineManager	
Manages virtual machines	62
VirtualMachineManagerMockGetInfoAboutVirtualMachine	65
VirtualMachineManagerTests	67
VirtualNetworkManager	
Manages virtual networks	67

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

src/exceptions/ PacketSnifferException.h	71
src/exceptions/ VirtualizationException.h	71
src/interfaces/ ILibpcapWrapper.h	71
src/interfaces/ ILibvirtWrapper.h	72
src/models/ ConnectionInfo.h	73
src/models/ ExecutionInfo.h	73
src/models/ ListenCallbackArgs.h	73
src/models/ NetworkDefinition.h	73
src/models/ Packet.h	74
src/models/ StreamData.h	74
src/models/ Version.h	74
src/models/ VirtualMachineInfo.h	74
src/packetsniffer/ PacketSniffer.h	75
src/utils/ ExecutionInfoObtainer.h	75
src/utils/ StringUtils.h	75
src/utils/ VersionUtils.h	75
src/virt/ VirtualizationFacade.h	78
src/virt/managers/ BaseManagerWithConnection.h	76
src/virt/managers/ ConnectionManager.h	76
src/virt/managers/ VirtualMachineConsoleManager.h	77
src/virt/managers/ VirtualMachineManager.h	77
src/virt/managers/ VirtualNetworkManager.h	77
src/wrappers/ LibpcapWrapper.h	79
src/wrappers/ LibvirtWrapper.h	79
tests/ TestingUtils.h	81
tests/mocks/ LibvirtWrapperMock.h	80
tests/mocks/ VirtualMachineManagerMockGetInfoAboutVirtualMachine.h	80

Chapter 4

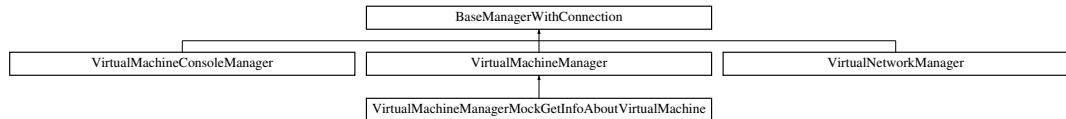
Class Documentation

4.1 BaseManagerWithConnection Class Reference

Base class for managers that require a libvirt connection.

```
#include <BaseManagerWithConnection.h>
```

Inheritance diagram for BaseManagerWithConnection:



Public Member Functions

- **BaseManagerWithConnection (ILibvirtWrapper *libvirt)**
Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.
- **BaseManagerWithConnection ()**
Constructs a [BaseManagerWithConnection](#) with default libvirt wrapper.
- void [updateConnection \(virConnectPtr conn\)](#)
Updates the libvirt connection pointer.

Protected Member Functions

- void [checkIfConnectionIsSet \(\) const](#)
Checks if the connection is set and throws an exception if not.

Protected Attributes

- [ILibvirtWrapper * libvirt](#)
- virConnectPtr [conn](#)

4.1.1 Detailed Description

Base class for managers that require a libvirt connection.

This abstract class provides common functionality for managers that need access to a libvirt connection and wrapper. It serves as a foundation for specific manager classes like [VirtualMachineManager](#) and [VirtualNetworkManager](#).

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BaseManagerWithConnection()

```
BaseManagerWithConnection::BaseManagerWithConnection (
    ILibvirtWrapper * libvirt) [inline], [explicit]
```

Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.

Parameters

<i>libvirt</i>	Pointer to an ILibvirtWrapper implementation.
----------------	---------------------------------------------------------------

4.1.3 Member Function Documentation

4.1.3.1 updateConnection()

```
void BaseManagerWithConnection::updateConnection (
    virConnectPtr conn)
```

Updates the libvirt connection pointer.

Parameters

<i>conn</i>	New connection pointer to use for operations.
-------------	-----------------------------------------------

The documentation for this class was generated from the following files:

- src/virt/managers/BaseManagerWithConnection.h
- src/virt/managers/BaseManagerWithConnection.cpp

4.2 ConnectionInfo Struct Reference

Structure containing information about a hypervisor connection.

```
#include <ConnectionInfo.h>
```

Public Attributes

- unsigned int **cpuCount**
Number of CPUs available on the host.
- unsigned int **cpuFreq**
CPU frequency in MHz.
- unsigned long **totalMemory**
Total memory available on the host in bytes.
- char **connectionUrl** [256]
Connection URL to the hypervisor.
- char **driverType** [256]
Type of the hypervisor driver (e.g., QEMU, KVM).
- **Version libVersion**
Version of the libvirt library.
- **Version driverVersion**
Version of the hypervisor driver.

4.2.1 Detailed Description

Structure containing information about a hypervisor connection.

This structure holds detailed information about the connection to a libvirt hypervisor, including hardware resources and version information.

The documentation for this struct was generated from the following file:

- src/models/ConnectionInfo.h

4.3 ConnectionManager Class Reference

Manages connections to libvirt hypervisors.

```
#include <ConnectionManager.h>
```

Public Member Functions

- **ConnectionManager (ILibvirtWrapper *libvirt)**
Constructs a [ConnectionManager](#) with a specific libvirt wrapper.
- **ConnectionManager ()**
Constructs a [ConnectionManager](#) with default libvirt wrapper.
- void **initializeConnection (const std::optional< std::string > &customConnectionUrl=std::nullopt)**
Initializes the connection to the libvirt backend.
- **ConnectionInfo getConnectionInfo () const**
Retrieves detailed information about the current connection.
- bool **isConnectionAlive () const**
Checks if the connection is alive and active.
- virConnectPtr **getConnection () const**
Gets the raw libvirt connection pointer.

4.3.1 Detailed Description

Manages connections to libvirt hypervisors.

This class handles the lifecycle of connections to libvirt, including initialization, status checking, and retrieving connection information.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ConnectionManager()

```
ConnectionManager::ConnectionManager (
    ILibvirtWrapper * libvirt) [inline], [explicit]
```

Constructs a [ConnectionManager](#) with a specific libvirt wrapper.

Parameters

<i>libvirt</i>	Pointer to an ILibvirtWrapper implementation.
----------------	---------------------------------------------------------------

4.3.3 Member Function Documentation

4.3.3.1 getConnection()

```
virConnectPtr ConnectionManager::getConnection () const
```

Gets the raw libvirt connection pointer.

Returns

`virConnectPtr` Pointer to the libvirt connection.

4.3.3.2 getConnectionInfo()

```
ConnectionInfo ConnectionManager::getConnectionInfo () const
```

Retrieves detailed information about the current connection.

Returns

`ConnectionInfo` Structure containing connection and host information.

4.3.3.3 initializeConnection()

```
void ConnectionManager::initializeConnection (
    const std::optional< std::string > & customConnectionString = std::nullopt)
```

Initializes the connection to the libvirt backend.

Parameters

<i>customConnectionUrl</i>	Optional URL for a custom hypervisor connection. If not provided, default connection is used.
----------------------------	-----------------------------------------------------------------------------------------------

4.3.3.4 isConnectionAlive()

```
bool ConnectionManager::isConnectionAlive () const
```

Checks if the connection is alive and active.

Returns

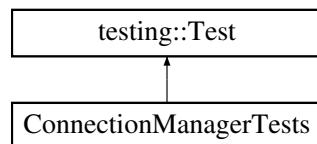
```
bool True if connection is alive, false otherwise.
```

The documentation for this class was generated from the following files:

- src/virt/managers/ConnectionManager.h
- src/virt/managers/ConnectionManager.cpp

4.4 ConnectionManagerTests Class Reference

Inheritance diagram for ConnectionManagerTests:

**Protected Attributes**

- [LibvirtWrapperMock](#) **mockLibvirt**
- [ConnectionManager](#) **manager**

The documentation for this class was generated from the following file:

- tests/ConnectionManagerTests.cpp

4.5 ExecutionInfo Struct Reference

Structure containing execution result information.

```
#include <ExecutionInfo.h>
```

Public Attributes

- bool **errorOccurred**
Flag indicating whether an error occurred during execution.
- char **msg** [128]
Error or status message describing the execution result.

4.5.1 Detailed Description

Structure containing execution result information.

This structure is used to return status and error information from function executions.

The documentation for this struct was generated from the following file:

- src/models/ExecutionInfo.h

4.6 ExecutionInfoObtainer Class Reference

Utility class for executing functions and capturing execution information.

```
#include <ExecutionInfoObtainer.h>
```

Static Public Member Functions

- static void [runAndObtainExecutionInfo](#) ([ExecutionInfo](#) *executionInfo, const std::function< void()> &func)
Executes a function and captures execution result information.

4.6.1 Detailed Description

Utility class for executing functions and capturing execution information.

This class provides a wrapper for running functions while automatically capturing any exceptions and converting them to [ExecutionInfo](#) structures.

4.6.2 Member Function Documentation

4.6.2.1 runAndObtainExecutionInfo()

```
void ExecutionInfoObtainer::runAndObtainExecutionInfo (
    ExecutionInfo * executionInfo,
    const std::function< void()> & func) [static]
```

Executes a function and captures execution result information.

This method runs the provided function and populates an [ExecutionInfo](#) structure with the result. If the function throws an exception, the error flag is set and the exception message is captured.

Parameters

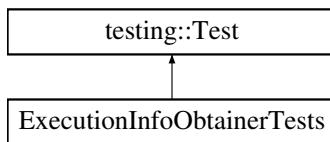
<code>executionInfo</code>	Pointer to ExecutionInfo structure to be filled with results.
<code>func</code>	Function to execute.

The documentation for this class was generated from the following files:

- src/utils/ExecutionInfoObtainer.h
- src/utils/ExecutionInfoObtainer.cpp

4.7 ExecutionInfoObtainerTests Class Reference

Inheritance diagram for ExecutionInfoObtainerTests:



The documentation for this class was generated from the following file:

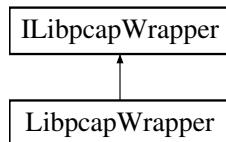
- tests/utils/ExecutionInfoObtainerTests.cpp

4.8 ILibpcapWrapper Class Reference

Interface for libpcap wrapper functionality.

```
#include <ILibpcapWrapper.h>
```

Inheritance diagram for ILibpcapWrapper:



Public Member Functions

- virtual `pcap_t * openHandlerLive` (`const std::string &interfaceName, char *errBuff)=0`
Opens a live packet capture handler for a network interface.
- virtual `int getLinkLayerType` (`pcap_t *handler)=0`
Gets the link-layer header type for a capture handler.
- virtual `void closeHandler` (`pcap_t *handler)=0`
Closes a packet capture handler.
- virtual `int listenForPackets` (`pcap_t *handler, pcap_handler callback, u_char *args)=0`
Starts listening for packets on the capture handler.
- virtual `void close` (`pcap_t *handler)=0`
Closes the packet capture handler and releases resources.

4.8.1 Detailed Description

Interface for libpcap wrapper functionality.

This abstract class defines an interface for interacting with the libpcap library for packet capture operations.

4.8.2 Member Function Documentation

4.8.2.1 close()

```
virtual void ILibpcapWrapper::close (
    pcap_t * handler) [pure virtual]
```

Closes the packet capture handler and releases resources.

Parameters

<i>handler</i>	Packet capture handler to close.
----------------	----------------------------------

Implemented in [LibpcapWrapper](#).

4.8.2.2 closeHandler()

```
virtual void ILibpcapWrapper::closeHandler (
    pcap_t * handler) [pure virtual]
```

Closes a packet capture handler.

Parameters

<i>handler</i>	Packet capture handler to close.
----------------	----------------------------------

Implemented in [LibpcapWrapper](#).

4.8.2.3 getLinkLayerType()

```
virtual int ILibpcapWrapper::getLinkLayerType (
    pcap_t * handler) [pure virtual]
```

Gets the link-layer header type for a capture handler.

Parameters

<i>handler</i>	Active packet capture handler.
----------------	--------------------------------

Implemented in [LibpcapWrapper](#).

4.8.2.4 listenForPackets()

```
virtual int ILibpcapWrapper::listenForPackets (
    pcap_t * handler,
    pcap_handler callback,
    u_char * args) [pure virtual]
```

Starts listening for packets on the capture handler.

Parameters

<i>handler</i>	Active packet capture handler.
<i>callback</i>	Function to call when a packet is captured.
<i>args</i>	User-defined arguments to pass to the callback function.

Implemented in [LibpcapWrapper](#).

4.8.2.5 openHandlerLive()

```
virtual pcap_t * ILibpcapWrapper::openHandlerLive (
    const std::string & interfaceName,
    char * errBuff) [pure virtual]
```

Opens a live packet capture handler for a network interface.

Parameters

<i>interfaceName</i>	Name of the network interface to capture packets from.
<i>errBuff</i>	Buffer to store error messages.

Returns

`pcap_t*` Pointer to the packet capture handler, or `nullptr` on failure.

Implemented in [LibpcapWrapper](#).

The documentation for this class was generated from the following file:

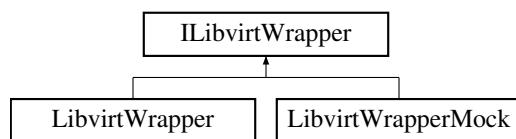
- src/interfaces/ILibpcapWrapper.h

4.9 ILibvirtWrapper Class Reference

Interface for Libvirt wrapper functionality.

```
#include <ILibvirtWrapper.h>
```

Inheritance diagram for ILibvirtWrapper:



Public Member Functions

- virtual virConnectPtr [connectOpen](#) (const char *name)=0
Opens a connection to a hypervisor.
- virtual virDomainPtr [createVirtualMachineFromXml](#) (virConnectPtr conn, const char *xmlConfig)=0
Creates a virtual machine from XML configuration.
- virtual void [getUuidFromDomain](#) (virDomainPtr domain, char *uuid)=0
Retrieves the UUID of a domain.
- virtual int [getNodeInfo](#) (virConnectPtr conn, virNodeInfoPtr info)=0
Gets information about the physical host node.
- virtual int [getLibVersion](#) (virConnectPtr conn, unsigned long *version)=0
Gets the version of the libvirt library.
- virtual int [getDriverVersion](#) (virConnectPtr conn, unsigned long *version)=0
Gets the version of the hypervisor driver.
- virtual std::string [getConnectUrl](#) (virConnectPtr conn)=0
Gets the connection URI.
- virtual std::string [getDriverType](#) (virConnectPtr conn)=0
Gets the type of hypervisor driver.
- virtual std::string [getLastErrorMessage](#) ()=0
Gets the last error message from libvirt.
- virtual virDomainPtr [domainLookupByName](#) (virConnectPtr conn, std::string name)=0
Looks up a domain by its name.
- virtual int [domainGetInfo](#) (virDomainPtr domain, virDomainInfo &domainInfo)=0
Gets information about a domain.
- virtual int [getDomainUUID](#) (virDomainPtr domain, std::string &uuid)=0
Gets the UUID of a domain.
- virtual int [getListOfTypeAllDomains](#) (virConnectPtr conn, virDomainPtr **domains)=0
Gets a list of all domains on the connection.
- virtual std::string [getDomainName](#) (virDomainPtr domain)=0
Gets the name of a domain.
- virtual void [freeDomain](#) (virDomainPtr domain)=0
Frees a domain handle.
- virtual int [connectionIsAlive](#) (virConnectPtr conn)=0
Checks if a connection is alive.
- virtual virStreamPtr [createNewStream](#) (virConnectPtr conn)=0
Creates a new stream object.
- virtual int [openDomainConsole](#) (virDomainPtr domain, virStreamPtr stream)=0
Opens a console connection to a domain.
- virtual virDomainPtr [domainLookupByUuid](#) (virConnectPtr conn, const std::string &uuid)=0
Looks up a domain by its UUID.
- virtual int [receiveDataFromStream](#) (virStreamPtr stream, char *buffer, int bufferSize)=0
Receives data from a stream.
- virtual void [sendDataToStream](#) (virStreamPtr stream, const char *buffer, int bufferSize)=0
Sends data to a stream.
- virtual void [finishAndFreeStream](#) (virStreamPtr stream)=0
Finishes and frees a stream.
- virtual virNetworkPtr [createNetworkFromXml](#) (virConnectPtr conn, const std::string &networkDefinition)=0
Creates a virtual network from XML definition.
- virtual int [attachDeviceToVm](#) (virDomainPtr domain, const std::string &deviceDefinition)=0
Attaches a device to a virtual machine.
- virtual int [detachDeviceFromVm](#) (virDomainPtr domain, const std::string &deviceDefinition)=0

- virtual int [updateVmDevice](#) (virDomainPtr domain, const std::string &deviceDefinition)=0

Detaches a device from a virtual machine.
- virtual virNetworkPtr [getNetworkByName](#) (virConnectPtr conn, const std::string &name)=0

Updates a device configuration in a virtual machine.
- virtual int [destroyNetwork](#) (virNetworkPtr network)=0

Gets a network by its name.
- virtual std::string [getNetworkDefinition](#) (virNetworkPtr network)=0

Destroys (stops and removes) a virtual network.
- virtual std::string [getNetworkDefinition](#) (virNetworkPtr network)=0

Gets the XML definition of a network.

4.9.1 Detailed Description

Interface for Libvirt wrapper functionality.

This abstract class defines an interface for interacting with the libvirt API.

4.9.2 Member Function Documentation

4.9.2.1 attachDeviceToVm()

```
virtual int ILibvirtWrapper::attachDeviceToVm (
    virDomainPtr domain,
    const std::string & deviceDefinition) [pure virtual]
```

Attaches a device to a virtual machine.

Parameters

<i>domain</i>	Domain to attach device to.
<i>deviceDefinition</i>	XML string defining the device.

Returns

int 0 on success, -1 on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.2 connectionIsAlive()

```
virtual int ILibvirtWrapper::connectionIsAlive (
    virConnectPtr conn) [pure virtual]
```

Checks if a connection is alive.

Parameters

<i>conn</i>	Connection to check.
-------------	----------------------

Returns

int 1 if alive, 0 if not, -1 on error.

Implemented in [LibvirtWrapper](#).

4.9.2.3 connectOpen()

```
virtual virConnectPtr ILibvirtWrapper::connectOpen (
    const char * name) [pure virtual]
```

Opens a connection to a hypervisor.

Parameters

<i>name</i>	URI or name of the hypervisor connection.
-------------	-------------------------------------------

Returns

`virConnectPtr` Connection handle, or `nullptr` on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.4 createNetworkFromXml()

```
virtual virNetworkPtr ILibvirtWrapper::createNetworkFromXml (
    virConnectPtr conn,
    const std::string & networkDefinition) [pure virtual]
```

Creates a virtual network from XML definition.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>networkDefinition</i>	XML string defining the network.

Returns

`virNetworkPtr` Network handle, or `nullptr` on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.5 createNewStream()

```
virtual virStreamPtr ILibvirtWrapper::createNewStream (
    virConnectPtr conn) [pure virtual]
```

Creates a new stream object.

Parameters

<i>conn</i>	Active connection to the hypervisor.
-------------	--------------------------------------

Returns

`virStreamPtr` Stream handle, or `nullptr` on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.6 createVirtualMachineFromXml()

```
virtual virDomainPtr ILibvirtWrapper::createVirtualMachineFromXml (
    virConnectPtr conn,
    const char * xmlConfig) [pure virtual]
```

Creates a virtual machine from XML configuration.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>xmlConfig</i>	XML string describing the VM configuration.

Returns

`virDomainPtr` Domain handle for the created VM.

Implemented in [LibvirtWrapper](#).

4.9.2.7 destroyNetwork()

```
virtual int ILibvirtWrapper::destroyNetwork (
    virNetworkPtr network) [pure virtual]
```

Destroys (stops and removes) a virtual network.

Parameters

<i>network</i>	Network to destroy.
----------------	---------------------

Returns

`int` 0 on success, -1 on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.8 detachDeviceFromVm()

```
virtual int ILibvirtWrapper::detachDeviceFromVm (
    virDomainPtr domain,
    const std::string & deviceDefinition) [pure virtual]
```

Detaches a device from a virtual machine.

Parameters

<i>domain</i>	Domain to detach device from.
<i>deviceDefinition</i>	XML string defining the device.

Returns

`int` 0 on success, -1 on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.9 domainGetInfo()

```
virtual int ILibvirtWrapper::domainGetInfo (
    virDomainPtr domain,
    virDomainInfo & domainInfo) [pure virtual]
```

Gets information about a domain.

Parameters

<i>domain</i>	Domain to get information from.
<i>domainInfo</i>	Reference to structure to be filled with domain information.

Returns

int 0 on success, -1 on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.10 domainLookupByName()

```
virtual virDomainPtr ILibvirtWrapper::domainLookupByName (
    virConnectPtr conn,
    std::string name) [pure virtual]
```

Looks up a domain by its name.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>name</i>	Name of the domain to look up.

Returns

virDomainPtr Domain handle, or nullptr if not found.

Implemented in [LibvirtWrapper](#).

4.9.2.11 domainLookupByUuid()

```
virtual virDomainPtr ILibvirtWrapper::domainLookupByUuid (
    virConnectPtr conn,
    const std::string & uuid) [pure virtual]
```

Looks up a domain by its UUID.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>uuid</i>	UUID string of the domain to find.

Returns

virDomainPtr Domain handle, or nullptr if not found.

Implemented in [LibvirtWrapper](#).

4.9.2.12 finishAndFreeStream()

```
virtual void ILibvirtWrapper::finishAndFreeStream (
    virStreamPtr stream) [pure virtual]
```

Finishes and frees a stream.

Parameters

<i>stream</i>	Stream to finish and free.
---------------	----------------------------

Implemented in [LibvirtWrapper](#).

4.9.2.13 freeDomain()

```
virtual void ILibvirtWrapper::freeDomain (
    virDomainPtr domain) [pure virtual]
```

Frees a domain handle.

Parameters

<i>domain</i>	Domain handle to free.
---------------	------------------------

Implemented in [LibvirtWrapper](#).

4.9.2.14 getConnectUrl()

```
virtual std::string ILibvirtWrapper::getConnectUrl (
    virConnectPtr conn) [pure virtual]
```

Gets the connection URI.

Parameters

<i>conn</i>	Active connection to the hypervisor.
-------------	--------------------------------------

Returns

`std::string` Connection URI string.

Implemented in [LibvirtWrapper](#).

4.9.2.15 getDomainName()

```
virtual std::string ILibvirtWrapper::getDomainName (
    virDomainPtr domain) [pure virtual]
```

Gets the name of a domain.

Parameters

<i>domain</i>	Domain to get name from.
---------------	--------------------------

Returns

`std::string` Domain name.

Implemented in [LibvirtWrapper](#).

4.9.2.16 getDomainUUID()

```
virtual int ILibvirtWrapper::getDomainUUID (
    virDomainPtr domain,
    std::string & uuid) [pure virtual]
```

Gets the UUID of a domain.

Parameters

<i>domain</i>	Domain to get UUID from.
<i>uuid</i>	Reference to string to store the UUID.

Returns

int 0 on success, -1 on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.17 getDriverType()

```
virtual std::string ILibvirtWrapper::getDriverType (
    virConnectPtr conn) [pure virtual]
```

Gets the type of hypervisor driver.

Parameters

<i>conn</i>	Active connection to the hypervisor.
-------------	--------------------------------------

Returns

std::string Driver type (e.g., "QEMU", "KVM").

Implemented in [LibvirtWrapper](#).

4.9.2.18 getDriverVersion()

```
virtual int ILibvirtWrapper::getDriverVersion (
    virConnectPtr conn,
    unsigned long * version) [pure virtual]
```

Gets the version of the hypervisor driver.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>version</i>	Pointer to store the driver version number.

Returns

int 0 on success, -1 on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.19 getLastErrorMessage()

```
virtual std::string ILibvirtWrapper::getLastErrorMessage () [pure virtual]
```

Gets the last error message from libvirt.

Returns

std::string Last error message.

Implemented in [LibvirtWrapper](#).

4.9.2.20 getLibVersion()

```
virtual int ILibvirtWrapper::getLibVersion (
    virConnectPtr conn,
    unsigned long * version) [pure virtual]
```

Gets the version of the libvirt library.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>version</i>	Pointer to store the version number.

Returns

int 0 on success, -1 on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.21 getListOfAllDomains()

```
virtual int ILibvirtWrapper::listOfAllDomains (
    virConnectPtr conn,
    virDomainPtr ** domains) [pure virtual]
```

Gets a list of all domains on the connection.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>domains</i>	Pointer to array that will be allocated and filled with domain handles.

Returns

int Number of domains found, or -1 on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.22 getNetworkByName()

```
virtual virNetworkPtr ILibvirtWrapper::getNetworkByName (
    virConnectPtr conn,
    const std::string & name) [pure virtual]
```

Gets a network by its name.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>name</i>	Name of the network to find.

Returns

`virNetworkPtr` Network handle, or `nullptr` if not found.

Implemented in [LibvirtWrapper](#).

4.9.2.23 getNetworkDefinition()

```
virtual std::string ILibvirtWrapper::getNetworkDefinition (
    virNetworkPtr network) [pure virtual]
```

Gets the XML definition of a network.

Parameters

<i>network</i>	Network to get definition from.
----------------	---------------------------------

Returns

`std::string` XML definition of the network.

Implemented in [LibvirtWrapper](#).

4.9.2.24 getNodeInfo()

```
virtual int ILibvirtWrapper::getNodeInfo (
    virConnectPtr conn,
    virNodeInfoPtr info) [pure virtual]
```

Gets information about the physical host node.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>info</i>	Pointer to structure to be filled with node information.

Returns

`int` 0 on success, -1 on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.25 getUuidFromDomain()

```
virtual void ILibvirtWrapper::getUuidFromDomain (
    virDomainPtr domain,
    char * uuid) [pure virtual]
```

Retrieves the UUID of a domain.

Parameters

<i>domain</i>	Domain to get UUID from.
<i>uuid</i>	Buffer to store the UUID string (must be at least VIR_UUID_STRING_BUflen bytes).

Implemented in [LibvirtWrapper](#).

4.9.2.26 openDomainConsole()

```
virtual int ILibvirtWrapper::openDomainConsole (
    virDomainPtr domain,
    virStreamPtr stream) [pure virtual]
```

Opens a console connection to a domain.

Parameters

<i>domain</i>	Domain to open console for.
<i>stream</i>	Stream to use for console I/O.

Returns

int 0 on success, -1 on failure.

Implemented in [LibvirtWrapper](#).

4.9.2.27 receiveDataFromStream()

```
virtual int ILibvirtWrapper::receiveDataFromStream (
    virStreamPtr stream,
    char * buffer,
    int bufferSize) [pure virtual]
```

Receives data from a stream.

Parameters

<i>stream</i>	Active stream to read from.
<i>buffer</i>	Buffer to store received data.
<i>bufferSize</i>	Size of the buffer.

Returns

int Number of bytes read, 0 on EOF, -1 on error.

Implemented in [LibvirtWrapper](#).

4.9.2.28 sendDataToStream()

```
virtual void ILibvirtWrapper::sendDataToStream (
    virStreamPtr stream,
    const char * buffer,
    int bufferSize) [pure virtual]
```

Sends data to a stream.

Parameters

<i>stream</i>	Active stream to write to.
<i>buffer</i>	Data to send.
<i>bufferSize</i>	Size of data to send.

Implemented in [LibvirtWrapper](#).

4.9.2.29 updateVmDevice()

```
virtual int ILibvirtWrapper::updateVmDevice (
    virDomainPtr domain,
    const std::string & deviceDefinition) [pure virtual]
```

Updates a device configuration in a virtual machine.

Parameters

<i>domain</i>	Domain to update device in.
<i>deviceDefinition</i>	XML string defining the new device configuration.

Returns

int 0 on success, -1 on failure.

Implemented in [LibvirtWrapper](#).

The documentation for this class was generated from the following file:

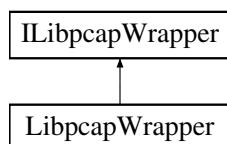
- src/interfaces/ILibvirtWrapper.h

4.10 LibpcapWrapper Class Reference

Concrete implementation of the [ILibpcapWrapper](#) interface.

```
#include <LibpcapWrapper.h>
```

Inheritance diagram for LibpcapWrapper:



Public Member Functions

- `pcap_t * openHandlerLive (const std::string &interfaceName, char *errBuff) override`
Opens a live packet capture handler for a network interface.
- `int getLinkLayerType (pcap_t *handler) override`
Gets the link-layer header type for a capture handler.
- `void closeHandler (pcap_t *handler) override`
Closes a packet capture handler.
- `int listenForPackets (pcap_t *handler, pcap_handler callback, u_char *args) override`
Starts listening for packets on the capture handler.
- `void close (pcap_t *handler) override`
Closes the packet capture handler and releases resources.

4.10.1 Detailed Description

Concrete implementation of the [ILibpcapWrapper](#) interface.

This class provides actual implementations for packet capture operations using the libpcap library.

4.10.2 Member Function Documentation

4.10.2.1 `close()`

```
void LibpcapWrapper::close (
    pcap_t * handler) [override], [virtual]
```

Closes the packet capture handler and releases resources.

Parameters

<code>handler</code>	Packet capture handler to close.
----------------------	----------------------------------

Implements [ILibpcapWrapper](#).

4.10.2.2 `closeHandler()`

```
void LibpcapWrapper::closeHandler (
    pcap_t * handler) [override], [virtual]
```

Closes a packet capture handler.

Parameters

<code>handler</code>	Packet capture handler to close.
----------------------	----------------------------------

Implements [ILibpcapWrapper](#).

4.10.2.3 `getLinkLayerType()`

```
int LibpcapWrapper::getLinkLayerType (
    pcap_t * handler) [override], [virtual]
```

Gets the link-layer header type for a capture handler.

Parameters

<i>handler</i>	Active packet capture handler.
----------------	--------------------------------

Returns

int Link-layer header type identifier.

Implements [ILibpcapWrapper](#).

4.10.2.4 listenForPackets()

```
int LibpcapWrapper::listenForPackets (
    pcap_t * handler,
    pcap_handler callback,
    u_char * args) [override], [virtual]
```

Starts listening for packets on the capture handler.

Parameters

<i>handler</i>	Active packet capture handler.
<i>callback</i>	Function to call when a packet is captured.
<i>args</i>	User-defined arguments to pass to the callback function.

Returns

int Number of packets processed, or error code.

Implements [ILibpcapWrapper](#).

4.10.2.5 openHandlerLive()

```
pcap_t * LibpcapWrapper::openHandlerLive (
    const std::string & interfaceName,
    char * errBuff) [override], [virtual]
```

Opens a live packet capture handler for a network interface.

Parameters

<i>interfaceName</i>	Name of the network interface to capture packets from.
<i>errBuff</i>	Buffer to store error messages if opening fails.

Returns

pcap_t* Pointer to the packet capture handler, or nullptr on failure.

Implements [ILibpcapWrapper](#).

The documentation for this class was generated from the following files:

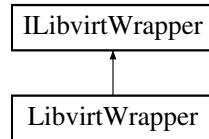
- src/wrappers/LibpcapWrapper.h
- src/wrappers/LibpcapWrapper.cpp

4.11 LibvirtWrapper Class Reference

Concrete implementation of the [ILibvirtWrapper](#) interface.

```
#include <LibvirtWrapper.h>
```

Inheritance diagram for LibvirtWrapper:



Public Member Functions

- `virConnectPtr connectOpen (const char *connectionUri) override`
Opens a connection to the hypervisor.
- `virDomainPtr createVirtualMachineFromXml (virConnectPtr conn, const char *xmlConfig) override`
Creates a virtual machine from an XML configuration.
- `void getUuidFromDomain (virDomainPtr domain, char *uuid) override`
Retrieves the UUID of the given domain.
- `int getNodeInfo (virConnectPtr conn, virNodeInfoPtr info) override`
Gets information about the physical host node.
- `int getLibVersion (virConnectPtr conn, unsigned long *libVersion) override`
Gets the version of the libvirt library.
- `int getDriverVersion (virConnectPtr conn, unsigned long *version) override`
Gets the version of the hypervisor driver.
- `std::string getConnectUrl (virConnectPtr conn) override`
Gets the connection URI.
- `std::string getDriverType (virConnectPtr conn) override`
Gets the type of hypervisor driver.
- `std::string getLastErrorMessage () override`
Gets the last error message from libvirt.
- `virDomainPtr domainLookupByName (virConnectPtr conn, std::string name) override`
Looks up a domain by its name.
- `int domainGetInfo (virDomainPtr domain, virDomainInfo &domainInfo) override`
Gets information about a domain.
- `int getDomainUUID (virDomainPtr domain, std::string &uuid) override`
Gets the UUID of a domain.
- `int getListOfAllDomains (virConnectPtr conn, virDomainPtr **domains) override`
Gets a list of all domains on the connection.
- `std::string getDomainName (virDomainPtr domain) override`
Gets the name of a domain.
- `void freeDomain (virDomainPtr domain) override`
Frees a domain handle.
- `int connectionIsAlive (virConnectPtr conn) override`
Checks if a connection is alive.
- `virStreamPtr createNewStream (virConnectPtr conn) override`
Creates a new stream object.

- int [openDomainConsole](#) (virDomainPtr domain, virStreamPtr stream) override
Opens a console connection to a domain.
- virDomainPtr [domainLookupByUuid](#) (virConnectPtr conn, const std::string &uuid) override
Looks up a domain by its UUID.
- int [receiveDataFromStream](#) (virStreamPtr stream, char *buffer, int bufferSize) override
Receives data from a stream.
- void [sendDataToStream](#) (virStreamPtr stream, const char *buffer, int bufferSize) override
Sends data to a stream.
- void [finishAndFreeStream](#) (virStreamPtr stream) override
Finishes and frees a stream.
- virNetworkPtr [createNetworkFromXml](#) (virConnectPtr conn, const std::string &networkDefinition) override
Creates a virtual network from XML definition.
- int [attachDeviceToVm](#) (virDomainPtr domain, const std::string &deviceDefinition) override
Attaches a device to a virtual machine.
- int [detachDeviceFromVm](#) (virDomainPtr domain, const std::string &deviceDefinition) override
Detaches a device from a virtual machine.
- int [updateVmDevice](#) (virDomainPtr domain, const std::string &deviceDefinition) override
Updates a device configuration in a virtual machine.
- virNetworkPtr [getNetworkByName](#) (virConnectPtr conn, const std::string &name) override
Gets a network by its name.
- int [destroyNetwork](#) (virNetworkPtr network) override
Destroys (stops and removes) a virtual network.
- std::string [getNetworkDefinition](#) (virNetworkPtr network) override
Gets the XML definition of a network.

4.11.1 Detailed Description

Concrete implementation of the [ILibvirtWrapper](#) interface.

This class provides actual implementations for connecting to the libvirt C API.

4.11.2 Member Function Documentation

4.11.2.1 attachDeviceToVm()

```
int LibvirtWrapper::attachDeviceToVm (
    virDomainPtr domain,
    const std::string & deviceDefinition) [override], [virtual]
```

Attaches a device to a virtual machine.

Parameters

<i>domain</i>	Domain to attach device to.
<i>deviceDefinition</i>	XML string defining the device.

Returns

int 0 on success, -1 on failure.

Implements [ILibvirtWrapper](#).

4.11.2.2 connectionIsAlive()

```
int LibvirtWrapper::connectionIsAlive (
    virConnectPtr conn) [override], [virtual]
```

Checks if a connection is alive.

Parameters

<i>conn</i>	Connection to check.
-------------	----------------------

Returns

int 1 if alive, 0 if not, -1 on error.

Implements [ILibvirtWrapper](#).

4.11.2.3 connectOpen()

```
virConnectPtr LibvirtWrapper::connectOpen (
    const char * connectionUri) [override], [virtual]
```

Opens a connection to the hypervisor.

Parameters

<i>connectionUri</i>	The name or URI of the hypervisor to connect to.
----------------------	--------------------------------------------------

Returns

virConnectPtr A pointer to the connection object on success.

Implements [ILibvirtWrapper](#).

4.11.2.4 createNetworkFromXml()

```
virNetworkPtr LibvirtWrapper::createNetworkFromXml (
    virConnectPtr conn,
    const std::string & networkDefinition) [override], [virtual]
```

Creates a virtual network from XML definition.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>networkDefinition</i>	XML string defining the network.

Returns

virNetworkPtr Network handle, or nullptr on failure.

Implements [ILibvirtWrapper](#).

4.11.2.5 createNewStream()

```
virStreamPtr LibvirtWrapper::createNewStream (
    virConnectPtr conn) [override], [virtual]
```

Creates a new stream object.

Parameters

<i>conn</i>	Active connection to the hypervisor.
-------------	--------------------------------------

Returns

`virStreamPtr` Stream handle, or `nullptr` on failure.

Implements [ILibvirtWrapper](#).

4.11.2.6 `createVirtualMachineFromXml()`

```
virDomainPtr LibvirtWrapper::createVirtualMachineFromXml (
    virConnectPtr conn,
    const char * xmlConfig) [override], [virtual]
```

Creates a virtual machine from an XML configuration.

Parameters

<i>conn</i>	The active connection to the hypervisor.
<i>xmlConfig</i>	The XML string describing the VM configuration.

Returns

`virDomainPtr` A pointer to the newly created virtual machine domain.

Implements [ILibvirtWrapper](#).

4.11.2.7 `destroyNetwork()`

```
int LibvirtWrapper::destroyNetwork (
    virNetworkPtr network) [override], [virtual]
```

Destroys (stops and removes) a virtual network.

Parameters

<i>network</i>	Network to destroy.
----------------	---------------------

Returns

int 0 on success, -1 on failure.

Implements [ILibvirtWrapper](#).

4.11.2.8 `detachDeviceFromVm()`

```
int LibvirtWrapper::detachDeviceFromVm (
    virDomainPtr domain,
    const std::string & deviceDefinition) [override], [virtual]
```

Detaches a device from a virtual machine.

Parameters

<i>domain</i>	Domain to detach device from.
<i>deviceDefinition</i>	XML string defining the device.

Returns

int 0 on success, -1 on failure.

Implements [ILibvirtWrapper](#).

4.11.2.9 domainGetInfo()

```
int LibvirtWrapper::domainGetInfo (
    virDomainPtr domain,
    virDomainInfo & domainInfo) [override], [virtual]
```

Gets information about a domain.

Parameters

<i>domain</i>	Domain to get information from.
<i>domainInfo</i>	Reference to structure to be filled with domain information.

Returns

int 0 on success, -1 on failure.

Implements [ILibvirtWrapper](#).

4.11.2.10 domainLookupByName()

```
virDomainPtr LibvirtWrapper::domainLookupByName (
    virConnectPtr conn,
    std::string name) [override], [virtual]
```

Looks up a domain by its name.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>name</i>	Name of the domain to look up.

Returns

virDomainPtr Domain handle, or nullptr if not found.

Implements [ILibvirtWrapper](#).

4.11.2.11 domainLookupByUuid()

```
virDomainPtr LibvirtWrapper::domainLookupByUuid (
    virConnectPtr conn,
    const std::string & uuid) [override], [virtual]
```

Looks up a domain by its UUID.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>uuid</i>	UUID string of the domain to find.

Returns

`virDomainPtr` Domain handle, or `nullptr` if not found.

Implements [ILibvirtWrapper](#).

4.11.2.12 `finishAndFreeStream()`

```
void LibvirtWrapper::finishAndFreeStream (
    virStreamPtr stream) [override], [virtual]
```

Finishes and frees a stream.

Parameters

<i>stream</i>	Stream to finish and free.
---------------	----------------------------

Implements [ILibvirtWrapper](#).

4.11.2.13 `freeDomain()`

```
void LibvirtWrapper::freeDomain (
    virDomainPtr domain) [override], [virtual]
```

Frees a domain handle.

Parameters

<i>domain</i>	Domain handle to free.
---------------	------------------------

Implements [ILibvirtWrapper](#).

4.11.2.14 `getConnectUrl()`

```
std::string LibvirtWrapper::getConnectUrl (
    virConnectPtr conn) [override], [virtual]
```

Gets the connection URI.

Parameters

<i>conn</i>	Active connection to the hypervisor.
-------------	--------------------------------------

Returns

`std::string` Connection URI string.

Implements [ILibvirtWrapper](#).

4.11.2.15 `getDomainName()`

```
std::string LibvirtWrapper::getDomainName (
    virDomainPtr domain) [override], [virtual]
```

Gets the name of a domain.

Parameters

<i>domain</i>	Domain to get name from.
---------------	--------------------------

Returns

std::string Domain name.

Implements [ILibvirtWrapper](#).

4.11.2.16 getDomainUUID()

```
int LibvirtWrapper::getDomainUUID (
    virDomainPtr domain,
    std::string & uuid) [override], [virtual]
```

Gets the UUID of a domain.

Parameters

<i>domain</i>	Domain to get UUID from.
<i>uuid</i>	Reference to string to store the UUID.

Returns

int 0 on success, -1 on failure.

Implements [ILibvirtWrapper](#).

4.11.2.17 getDriverType()

```
std::string LibvirtWrapper::getDriverType (
    virConnectPtr conn) [override], [virtual]
```

Gets the type of hypervisor driver.

Parameters

<i>conn</i>	Active connection to the hypervisor.
-------------	--------------------------------------

Returns

std::string Driver type (e.g., "QEMU", "KVM").

Implements [ILibvirtWrapper](#).

4.11.2.18 getDriverVersion()

```
int LibvirtWrapper::getDriverVersion (
    virConnectPtr conn,
    unsigned long * version) [override], [virtual]
```

Gets the version of the hypervisor driver.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>version</i>	Pointer to store the driver version number.

Returns

int 0 on success, -1 on failure.

Implements [ILibvirtWrapper](#).

4.11.2.19 getLastErrorMessage()

```
std::string LibvirtWrapper::getLastErrorMessage () [override], [virtual]
```

Gets the last error message from libvirt.

Returns

std::string Last error message.

Implements [ILibvirtWrapper](#).

4.11.2.20 getLibVersion()

```
int LibvirtWrapper::getLibVersion (
    virConnectPtr conn,
    unsigned long * libVersion) [override], [virtual]
```

Gets the version of the libvirt library.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>libVersion</i>	Pointer to store the version number.

Returns

int 0 on success, -1 on failure.

Implements [ILibvirtWrapper](#).

4.11.2.21 getListOfAllDomains()

```
int LibvirtWrapper::getListOfAllDomains (
    virConnectPtr conn,
    virDomainPtr ** domains) [override], [virtual]
```

Gets a list of all domains on the connection.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>domains</i>	Pointer to array that will be allocated and filled with domain handles.

Returns

int Number of domains found, or -1 on failure.

Implements [ILibvirtWrapper](#).

4.11.2.22 getNetworkByName()

```
virNetworkPtr LibvirtWrapper::getNetworkByName (
    virConnectPtr conn,
    const std::string & name) [override], [virtual]
```

Gets a network by its name.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>name</i>	Name of the network to find.

Returns

virNetworkPtr Network handle, or nullptr if not found.

Implements [ILibvirtWrapper](#).

4.11.2.23 getNetworkDefinition()

```
std::string LibvirtWrapper::getNetworkDefinition (
    virNetworkPtr network) [override], [virtual]
```

Gets the XML definition of a network.

Parameters

<i>network</i>	Network to get definition from.
----------------	---------------------------------

Returns

std::string XML definition of the network.

Implements [ILibvirtWrapper](#).

4.11.2.24 getNodeInfo()

```
int LibvirtWrapper::getNodeInfo (
    virConnectPtr conn,
    virNodeInfoPtr info) [override], [virtual]
```

Gets information about the physical host node.

Parameters

<i>conn</i>	Active connection to the hypervisor.
<i>info</i>	Pointer to structure to be filled with node information.

Returns

int 0 on success, -1 on failure.

Implements [ILibvirtWrapper](#).

4.11.2.25 getUuidFromDomain()

```
void LibvirtWrapper::getUuidFromDomain (
    virDomainPtr domain,
    char * uuid) [override], [virtual]
```

Retrieves the UUID of the given domain.

Parameters

<i>domain</i>	The domain to get the UUID from.
<i>uuid</i>	A buffer to store the resulting UUID string.

Implements [ILibvirtWrapper](#).

4.11.2.26 openDomainConsole()

```
int LibvirtWrapper::openDomainConsole (
    virDomainPtr domain,
    virStreamPtr stream) [override], [virtual]
```

Opens a console connection to a domain.

Parameters

<i>domain</i>	Domain to open console for.
<i>stream</i>	Stream to use for console I/O.

Returns

int 0 on success, -1 on failure.

Implements [ILibvirtWrapper](#).

4.11.2.27 receiveDataFromStream()

```
int LibvirtWrapper::receiveDataFromStream (
    virStreamPtr stream,
    char * buffer,
    int bufferSize) [override], [virtual]
```

Receives data from a stream.

Parameters

<i>stream</i>	Active stream to read from.
<i>buffer</i>	Buffer to store received data.
<i>bufferSize</i>	Size of the buffer.

Returns

int Number of bytes read, 0 on EOF, -1 on error.

Implements [ILibvirtWrapper](#).

4.11.2.28 sendDataStream()

```
void LibvirtWrapper::sendDataStream (
    virStreamPtr stream,
    const char * buffer,
    int bufferSize) [override], [virtual]
```

Sends data to a stream.

Parameters

<i>stream</i>	Active stream to write to.
<i>buffer</i>	Data to send.
<i>bufferSize</i>	Size of data to send.

Implements [ILibvirtWrapper](#).

4.11.2.29 updateVmDevice()

```
int LibvirtWrapper::updateVmDevice (
    virDomainPtr domain,
    const std::string & deviceDefinition) [override], [virtual]
```

Updates a device configuration in a virtual machine.

Parameters

<i>domain</i>	Domain to update device in.
<i>deviceDefinition</i>	XML string defining the new device configuration.

Returns

int 0 on success, -1 on failure.

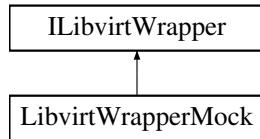
Implements [ILibvirtWrapper](#).

The documentation for this class was generated from the following files:

- src/wrappers/LibvirtWrapper.h
- src/wrappers/LibvirtWrapper.cpp

4.12 LibvirtWrapperMock Class Reference

Inheritance diagram for LibvirtWrapperMock:



Public Member Functions

- **MOCK_METHOD** (`virConnectPtr, connectOpen,(const char *name),(override)`)
- **MOCK_METHOD** (`virDomainPtr, createVirtualMachineFromXml,(virConnectPtr conn, const char *xmlConfig),(override)`)
- **MOCK_METHOD** (`void, getUuidFromDomain,(virDomainPtr domain, char *uuid),(override)`)
- **MOCK_METHOD** (`int, getNodeInfo,(virConnectPtr conn, virNodeInfoPtr info),(override)`)
- **MOCK_METHOD** (`int, getLibVersion,(virConnectPtr conn, unsigned long *version),(override)`)
- **MOCK_METHOD** (`int, getDriverVersion,(virConnectPtr conn, unsigned long *version),(override)`)
- **MOCK_METHOD** (`std::string, getConnectUrl,(virConnectPtr conn),(override)`)
- **MOCK_METHOD** (`std::string, getDriverType,(virConnectPtr conn),(override)`)
- **MOCK_METHOD** (`std::string, getLastErrorMessage,(),(override)`)
- **MOCK_METHOD** (`virDomainPtr, domainLookupByName,(virConnectPtr conn, std::string name),(override)`)
- **MOCK_METHOD** (`int, domainGetInfo,(virDomainPtr domain, virDomainInfo &domainInfo),(override)`)
- **MOCK_METHOD** (`int, getDomainUUID,(virDomainPtr domain, std::string &uuid),(override)`)
- **MOCK_METHOD** (`int, getListOfAllDomains,(virConnectPtr conn, virDomainPtr **domains),(override)`)
- **MOCK_METHOD** (`std::string, getDomainName,(virDomainPtr domain),(override)`)
- **MOCK_METHOD** (`void, freeDomain,(virDomainPtr domain),(override)`)
- **MOCK_METHOD** (`int, connectionIsAlive,(virConnectPtr conn),(override)`)
- **MOCK_METHOD** (`virStreamPtr, createNewStream,(virConnectPtr conn),(override)`)
- **MOCK_METHOD** (`int, openDomainConsole,(virDomainPtr domain, virStreamPtr stream),(override)`)
- **MOCK_METHOD** (`virDomainPtr, domainLookupByUuid,(virConnectPtr conn, const std::string &uuid),(override)`)
- **MOCK_METHOD** (`int, receiveDataFromStream,(virStreamPtr stream, char *buffer, int bufferSize),(override)`)
- **MOCK_METHOD** (`void, sendDataToStream,(virStreamPtr stream, const char *buffer, int bufferSize),(override)`)
- **MOCK_METHOD** (`void, finishAndFreeStream,(virStreamPtr stream),(override)`)
- **MOCK_METHOD** (`virNetworkPtr, createNetworkFromXml,(virConnectPtr conn, const std::string &networkDefinition),(override)`)
- **MOCK_METHOD** (`int, attachDeviceToVm,(virDomainPtr domain, const std::string &deviceDefinition),(override)`)
- **MOCK_METHOD** (`int, detachDeviceFromVm,(virDomainPtr domain, const std::string &deviceDefinition),(override)`)
- **MOCK_METHOD** (`int, updateVmDevice,(virDomainPtr domain, const std::string &deviceDefinition),(override)`)
- **MOCK_METHOD** (`virNetworkPtr, getNetworkByName,(virConnectPtr conn, const std::string &name),(override)`)
- **MOCK_METHOD** (`int, destroyNetwork,(virNetworkPtr network),(override)`)
- **MOCK_METHOD** (`std::string, getNetworkDefinition,(virNetworkPtr network),(override)`)

Public Member Functions inherited from [ILibvirtWrapper](#)

- virtual `virConnectPtr connectOpen (const char *name)=0`
Opens a connection to a hypervisor.
- virtual `virDomainPtr createVirtualMachineFromXml (virConnectPtr conn, const char *xmlConfig)=0`
Creates a virtual machine from XML configuration.
- virtual `void getUuidFromDomain (virDomainPtr domain, char *uuid)=0`
Retrieves the UUID of a domain.
- virtual `int getNodeInfo (virConnectPtr conn, virNodeInfoPtr info)=0`
Gets information about the physical host node.
- virtual `int getLibVersion (virConnectPtr conn, unsigned long *version)=0`
Gets the version of the libvirt library.
- virtual `int getDriverVersion (virConnectPtr conn, unsigned long *version)=0`
Gets the version of the hypervisor driver.
- virtual `std::string getConnectUrl (virConnectPtr conn)=0`
Gets the connection URI.
- virtual `std::string getDriverType (virConnectPtr conn)=0`
Gets the type of hypervisor driver.
- virtual `std::string getLastError ()=0`
Gets the last error message from libvirt.
- virtual `virDomainPtr domainLookupByName (virConnectPtr conn, std::string name)=0`
Looks up a domain by its name.
- virtual `int domainGetInfo (virDomainPtr domain, virDomainInfo &domainInfo)=0`
Gets information about a domain.
- virtual `int getDomainUUID (virDomainPtr domain, std::string &uuid)=0`
Gets the UUID of a domain.
- virtual `int getListOfAllDomains (virConnectPtr conn, virDomainPtr **domains)=0`
Gets a list of all domains on the connection.
- virtual `std::string getDomainName (virDomainPtr domain)=0`
Gets the name of a domain.
- virtual `void freeDomain (virDomainPtr domain)=0`
Frees a domain handle.
- virtual `int connectionIsAlive (virConnectPtr conn)=0`
Checks if a connection is alive.
- virtual `virStreamPtr createNewStream (virConnectPtr conn)=0`
Creates a new stream object.
- virtual `int openDomainConsole (virDomainPtr domain, virStreamPtr stream)=0`
Opens a console connection to a domain.
- virtual `virDomainPtr domainLookupByUuid (virConnectPtr conn, const std::string &uuid)=0`
Looks up a domain by its UUID.
- virtual `int receiveDataFromStream (virStreamPtr stream, char *buffer, int bufferSize)=0`
Receives data from a stream.
- virtual `void sendDataToStream (virStreamPtr stream, const char *buffer, int bufferSize)=0`
Sends data to a stream.
- virtual `void finishAndFreeStream (virStreamPtr stream)=0`
Finishes and frees a stream.
- virtual `virNetworkPtr createNetworkFromXml (virConnectPtr conn, const std::string &networkDefinition)=0`
Creates a virtual network from XML definition.
- virtual `int attachDeviceToVm (virDomainPtr domain, const std::string &deviceDefinition)=0`
Attaches a device to a virtual machine.
- virtual `int detachDeviceFromVm (virDomainPtr domain, const std::string &deviceDefinition)=0`

- Detaches a device from a virtual machine.*
- virtual int [updateVmDevice](#) (virDomainPtr domain, const std::string &deviceDefinition)=0
Updates a device configuration in a virtual machine.
 - virtual virNetworkPtr [getNetworkByName](#) (virConnectPtr conn, const std::string &name)=0
Gets a network by its name.
 - virtual int [destroyNetwork](#) (virNetworkPtr network)=0
Destroys (stops and removes) a virtual network.
 - virtual std::string [getNetworkDefinition](#) (virNetworkPtr network)=0
Gets the XML definition of a network.

The documentation for this class was generated from the following file:

- tests/mocks/LibvirtWrapperMock.h

4.13 ListenCallbackArgs Struct Reference

Structure containing arguments for packet listening callbacks.

```
#include <ListenCallbackArgs.h>
```

Public Attributes

- [PacketSniffer](#) * **sniffer**
Pointer to the [PacketSniffer](#) instance handling the capture.
- std::string **interfaceName**
Name of the network interface being monitored.

4.13.1 Detailed Description

Structure containing arguments for packet listening callbacks.

This structure is passed to packet capture callback functions to provide context about the packet sniffer and interface being monitored.

The documentation for this struct was generated from the following file:

- src/models/ListenCallbackArgs.h

4.14 NetworkDefinition Struct Reference

Structure containing network definition in XML format.

```
#include <NetworkDefinition.h>
```

Public Attributes

- char **content** [4096]
XML content defining the network configuration.

4.14.1 Detailed Description

Structure containing network definition in XML format.

This structure holds the XML configuration for defining virtual networks.

The documentation for this struct was generated from the following file:

- src/models/NetworkDefinition.h

4.15 Packet Struct Reference

Structure representing a captured network packet.

```
#include <Packet.h>
```

Public Attributes

- char **interfaceName** [64]
Name of the network interface where the packet was captured.
- unsigned char * **content**
Pointer to the raw packet data.
- int **contentLength**
Length of the packet content in bytes.
- unsigned int **timestampMicroseconds**
Timestamp of packet capture in microseconds.

4.15.1 Detailed Description

Structure representing a captured network packet.

This structure holds all the information about a single captured packet, including its content, length, timestamp, and the interface it was captured on.

The documentation for this struct was generated from the following file:

- src/models/Packet.h

4.16 PacketSniffer Class Reference

Class for capturing network packets.

```
#include <PacketSniffer.h>
```

Public Member Functions

- **PacketSniffer ()**
Constructs a [PacketSniffer](#) with default libpcap wrapper.
- **pcap_t * openSnifferHandler (const std::string &interfaceName)**
Opens a packet capture handler for the specified network interface.
- **bool listenForPacket (pcap_t *snifferHandler, const std::string &interfaceName)**
Listens for a single packet on the specified interface.
- **void closeAndStopListening (pcap_t *handler) const**
Closes the packet capture handler and stops listening.
- **Packet getPacketFromQueue ()**
Retrieves and removes a packet from the internal queue.
- **int getNumberOfReceivedPackets () const**
Gets the number of packets currently in the queue.

4.16.1 Detailed Description

Class for capturing network packets.

This class provides functionality for capturing packets from network interfaces using the libpcap library. It maintains a queue of captured packets and handles the packet capture lifecycle.

4.16.2 Member Function Documentation

4.16.2.1 closeAndStopListening()

```
void PacketSniffer::closeAndStopListening (
    pcap_t * handler) const
```

Closes the packet capture handler and stops listening.

Parameters

<i>handler</i>	Packet capture handler to close.
----------------	--------------------------------------------------

4.16.2.2 getNumberOfReceivedPackets()

```
int PacketSniffer::getNumberOfReceivedPackets () const [nodiscard]
```

Gets the number of packets currently in the queue.

Returns

int Number of captured packets waiting in the queue.

4.16.2.3 `getPacketFromQueue()`

```
Packet PacketSniffer::getPacketFromQueue () [nodiscard]
```

Retrieves and removes a packet from the internal queue.

Returns

`Packet` The oldest packet in the queue.

4.16.2.4 `listenForPacket()`

```
bool PacketSniffer::listenForPacket (
    pcap_t * snifferHandler,
    const std::string & interfaceName) [nodiscard]
```

Listens for a single packet on the specified interface.

This method captures one packet and adds it to the internal queue.

Parameters

<code>snifferHandler</code>	Active packet capture handler.
<code>interfaceName</code>	Name of the network interface.

Returns

`bool` True if a packet was successfully captured, false otherwise.

4.16.2.5 `openSnifferHandler()`

```
pcap_t * PacketSniffer::openSnifferHandler (
    const std::string & interfaceName) [nodiscard]
```

Opens a packet capture handler for the specified network interface.

Parameters

<code>interfaceName</code>	Name of the network interface to capture packets from.
----------------------------	--------------------------------------------------------

Returns

`pcap_t*` Pointer to the packet capture handler.

The documentation for this class was generated from the following files:

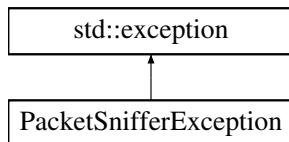
- src/packetsniffer/PacketSniffer.h
- src/packetsniffer/PacketSniffer.cpp

4.17 PacketSnifferException Class Reference

Exception class for packet sniffer-related errors.

```
#include <PacketSnifferException.h>
```

Inheritance diagram for PacketSnifferException:



Public Member Functions

- [PacketSnifferException](#) (std::string msg)
Constructs a [PacketSnifferException](#) with a descriptive message.
- const char * [what\(\)](#) const noexcept override
Returns the explanatory error message.

4.17.1 Detailed Description

Exception class for packet sniffer-related errors.

This exception is thrown when errors occur during packet capture operations.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 PacketSnifferException()

```
PacketSnifferException::PacketSnifferException (
    std::string msg) [inline], [explicit]
```

Constructs a [PacketSnifferException](#) with a descriptive message.

Parameters

<i>msg</i>	Error message describing the exception.
------------	-----------------------------------------

4.17.3 Member Function Documentation

4.17.3.1 what()

```
const char * PacketSnifferException::what () const [inline], [nodiscard], [override], [noexcept]
```

Returns the explanatory error message.

Returns

const char* Pointer to the error message string.

The documentation for this class was generated from the following file:

- src/exceptions/PacketSnifferException.h

4.18 StreamData Struct Reference

Structure containing data received from a virtual machine console stream.

```
#include <StreamData.h>
```

Public Attributes

- char **buffer** [255]
Buffer containing data read from the stream.
- bool **isStreamBroken**
Flag indicating whether the stream connection is broken.

4.18.1 Detailed Description

Structure containing data received from a virtual machine console stream.

This structure holds data read from a VM console stream along with the stream status.

The documentation for this struct was generated from the following file:

- src/models/StreamData.h

4.19 StringUtils Class Reference

Utility class for string manipulation operations.

```
#include <StringUtils.h>
```

Static Public Member Functions

- static void **copyStringToCharArray** (const std::string &src, char *charArray, int length)
Copies a std::string to a character array with specified length.

4.19.1 Detailed Description

Utility class for string manipulation operations.

This class provides static helper methods for converting and copying strings.

4.19.2 Member Function Documentation

4.19.2.1 copyStringToCharArray()

```
void StringUtils::copyStringToCharArray (
    const std::string & src,
    char * charArray,
    int length) [static]
```

Copies a std::string to a character array with specified length.

This method safely copies a string to a fixed-size character array, ensuring null-termination and preventing buffer overflow.

Parameters

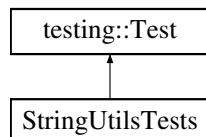
<i>src</i>	Source string to copy.
<i>charArray</i>	Destination character array.
<i>length</i>	Maximum length of the destination array including null terminator.

The documentation for this class was generated from the following files:

- src/utils/StringUtils.h
- src/utils/StringUtils.cpp

4.20 StringUtilsTests Class Reference

Inheritance diagram for StringUtilsTests:



The documentation for this class was generated from the following file:

- tests/utils/StringUtilsTests.cpp

4.21 TestingUtils Class Reference

Static Public Member Functions

- static void **expectThrowWithMessage** (const std::function< void()> &func, const std::string &expectedMessage)

The documentation for this class was generated from the following files:

- tests/TestingUtils.h
- tests/TestingUtils.cpp

4.22 Version Struct Reference

Structure representing a semantic version number.

```
#include <Version.h>
```

Public Attributes

- unsigned int **major**
Major version number.
- unsigned int **minor**
Minor version number.
- unsigned int **patch**
Patch version number.

4.22.1 Detailed Description

Structure representing a semantic version number.

This structure follows the semantic versioning convention (MAJOR.MINOR.PATCH).

The documentation for this struct was generated from the following file:

- src/models/Version.h

4.23 VersionUtils Class Reference

Utility class for version number operations.

```
#include <VersionUtils.h>
```

Static Public Member Functions

- static [Version getVersion](#) (unsigned long version)
Converts a packed version number to a [Version](#) structure.

4.23.1 Detailed Description

Utility class for version number operations.

This class provides static helper methods for parsing and converting version numbers.

4.23.2 Member Function Documentation

4.23.2.1 getVersion()

```
Version VersionUtils::getVersion (
    unsigned long version) [static]
```

Converts a packed version number to a [Version](#) structure.

This method decodes a libvirt-style packed version number (unsigned long) into separate major, minor, and patch components.

Parameters

<code>version</code>	Packed version number (e.g., from libvirt API).
----------------------	-------------------------------------------------

Returns

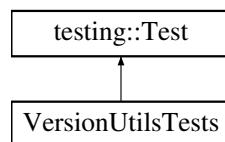
[Version](#) Structure containing major, minor, and patch version numbers.

The documentation for this class was generated from the following files:

- src/utils/VersionUtils.h
- src/utils/VersionUtils.cpp

4.24 VersionUtilsTests Class Reference

Inheritance diagram for VersionUtilsTests:



The documentation for this class was generated from the following file:

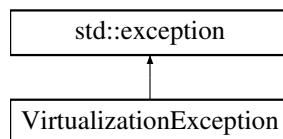
- tests/utils/VersionUtilsTests.cpp

4.25 VirtualizationException Class Reference

Exception class for virtualization-related errors.

```
#include <VirtualizationException.h>
```

Inheritance diagram for VirtualizationException:



Public Member Functions

- [VirtualizationException](#) (std::string msg)
Constructs a [VirtualizationException](#) with a descriptive message.
- const char * [what \(\)](#) const noexcept override
Returns the explanatory error message.

4.25.1 Detailed Description

Exception class for virtualization-related errors.

This exception is thrown when errors occur during virtualization operations

4.25.2 Constructor & Destructor Documentation

4.25.2.1 VirtualizationException()

```
VirtualizationException::VirtualizationException ( std::string msg) [inline], [explicit]
```

Constructs a [VirtualizationException](#) with a descriptive message.

Parameters

<i>msg</i>	Error message describing the exception.
------------	-----------------------------------------

4.25.3 Member Function Documentation

4.25.3.1 what()

```
const char * VirtualizationException::what () const [inline], [nodiscard], [override], [noexcept]
```

Returns the explanatory error message.

Returns

const char* Pointer to the error message string.

The documentation for this class was generated from the following file:

- src/exceptions/VirtualizationException.h

4.26 VirtualizationFacade Class Reference

Facade class providing a unified interface for virtualization operations.

```
#include <VirtualizationFacade.h>
```

Public Member Functions

- **VirtualizationFacade (ILibvirtWrapper *libvirt)**
Constructs a [VirtualizationFacade](#) with a specific libvirt wrapper.
- **VirtualizationFacade ()**
Constructs a [VirtualizationFacade](#) with default libvirt wrapper.
- void **initializeConnection (const char *customConnectionUrl) const**
Initializes the connection to the hypervisor.
- void **getConnectionString (ConnectionInfo *infoPtr) const**
Retrieves detailed information about the hypervisor connection.
- void **createVirtualMachine (const std::string &virtualMachineXml) const**
Creates a new virtual machine from XML definition.
- void **getInfoAboutVirtualMachine (VirtualMachineInfo *virtualMachineInfo, const std::string &name) const**
Retrieves information about a specific virtual machine.
- void **getListOfVirtualMachinesWithInfo (VirtualMachineInfo **arrayOfVirtualMachines, int *numberOfVirtualMachines) const**
Retrieves a list of all virtual machines with their information.
- void **isConnectionAlive (bool *isAlive) const**
Checks if the hypervisor connection is alive.
- virStreamPtr **openVirtualMachineConsole (const std::string &vmUuid) const**
Opens a console stream to a virtual machine.
- void **receiveDataFromConsole (virStreamPtr stream, StreamData *streamData) const**
Receives data from a virtual machine console.
- void **sendDataToConsole (virStreamPtr stream, const std::string &data) const**
Sends data to a virtual machine console.
- void **closeStream (virStreamPtr stream) const**
Closes a console stream.
- virNetworkPtr **createVirtualNetworkFromXml (const std::string &networkDefinition) const**
Creates a virtual network from XML definition.
- void **attachDeviceToVm (const std::string &uuid, const std::string &deviceDefinition) const**
Attaches a device to a virtual machine.
- void **detachDeviceFromVm (const std::string &uuid, const std::string &deviceDefinition) const**
Detaches a device from a virtual machine.
- void **updateVmDevice (const std::string &uuid, const std::string &deviceDefinition) const**
Updates a device configuration in a virtual machine.
- void **destroyNetwork (const std::string &name) const**
Destroys (stops and removes) a virtual network.
- std::string **getNetworkDefinition (const std::string &name) const**
Retrieves the XML definition of a virtual network.

4.26.1 Detailed Description

Facade class providing a unified interface for virtualization operations.

This class serves as a high-level interface that coordinates multiple manager classes to provide comprehensive virtualization functionality including connection management, virtual machine operations, console access, and network management.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 VirtualizationFacade()

```
VirtualizationFacade::VirtualizationFacade (
```

`ILibvirtWrapper * libvirt) [explicit]`

Constructs a [VirtualizationFacade](#) with a specific libvirt wrapper.

Parameters

<i>libvirt</i>	Pointer to an ILibvirtWrapper implementation.
----------------	---------------------------------------------------------------

4.26.3 Member Function Documentation

4.26.3.1 attachDeviceToVm()

```
void VirtualizationFacade::attachDeviceToVm (
    const std::string & uuid,
    const std::string & deviceDefinition) const
```

Attaches a device to a virtual machine.

Parameters

<i>uuid</i>	UUID of the virtual machine.
<i>deviceDefinition</i>	XML string defining the device to attach.

4.26.3.2 closeStream()

```
void VirtualizationFacade::closeStream (
    virStreamPtr stream) const
```

Closes a console stream.

Parameters

<i>stream</i>	Console stream to close.
---------------	--------------------------

4.26.3.3 createVirtualMachine()

```
void VirtualizationFacade::createVirtualMachine (
    const std::string & virtualMachineXml) const
```

Creates a new virtual machine from XML definition.

Parameters

<i>virtualMachineXml</i>	XML string defining the virtual machine configuration.
--------------------------	--------------------------------------------------------

4.26.3.4 createVirtualNetworkFromXml()

```
virNetworkPtr VirtualizationFacade::createVirtualNetworkFromXml (
    const std::string & networkDefinition) const [nodiscard]
```

Creates a virtual network from XML definition.

Parameters

<i>networkDefinition</i>	XML string defining the network configuration.
--------------------------	------------------------------------------------

Returns

`virNetworkPtr` Pointer to the created network.

4.26.3.5 `destroyNetwork()`

```
void VirtualizationFacade::destroyNetwork (
    const std::string & name) const
```

Destroys (stops and removes) a virtual network.

Parameters

<i>name</i>	Name of the network to destroy.
-------------	---------------------------------

4.26.3.6 `detachDeviceFromVm()`

```
void VirtualizationFacade::detachDeviceFromVm (
    const std::string & uuid,
    const std::string & deviceDefinition) const
```

Detaches a device from a virtual machine.

Parameters

<i>uuid</i>	UUID of the virtual machine.
<i>deviceDefinition</i>	XML string defining the device to detach.

4.26.3.7 `getConnectionString()`

```
void VirtualizationFacade::getConnectionString (
    ConnectionInfo * infoPtr) const
```

Retrieves detailed information about the hypervisor connection.

Parameters

<i>infoPtr</i>	Pointer to <code>ConnectionInfo</code> structure to be filled with connection data.
----------------	-------------------------------------------------------------------------------------

4.26.3.8 `getInfoAboutVirtualMachine()`

```
void VirtualizationFacade::getInfoAboutVirtualMachine (
    VirtualMachineInfo * virtualMachineInfo,
    const std::string & name) const
```

Retrieves information about a specific virtual machine.

Parameters

<code>virtualMachineInfo</code>	Pointer to <code>VirtualMachineInfo</code> structure to be filled.
<code>name</code>	Name of the virtual machine.

4.26.3.9 getListOfVirtualMachinesWithInfo()

```
void VirtualizationFacade::getListOfVirtualMachinesWithInfo (
    VirtualMachineInfo ** arrayOfVirtualMachines,
    int * numberOfVirtualMachines) const
```

Retrieves a list of all virtual machines with their information.

Parameters

<code>arrayOfVirtualMachines</code>	Pointer to array that will be allocated and filled with VM info.
<code>numberOfVirtualMachines</code>	Pointer to integer that will store the count of VMs.

4.26.3.10 getNetworkDefinition()

```
std::string VirtualizationFacade::getNetworkDefinition (
    const std::string & name) const [nodiscard]
```

Retrieves the XML definition of a virtual network.

Parameters

<code>name</code>	Name of the network.
-------------------	----------------------

Returns

`std::string` XML definition of the network.

4.26.3.11 initializeConnection()

```
void VirtualizationFacade::initializeConnection (
    const char * customConnectionString) const
```

Initializes the connection to the hypervisor.

Parameters

<code>customConnectionString</code>	Optional custom connection URL. If nullptr, default connection is used.
-------------------------------------	-------------------------------------------------------------------------

4.26.3.12 isConnectionAlive()

```
void VirtualizationFacade::isConnectionAlive (
    bool * isAlive) const
```

Checks if the hypervisor connection is alive.

Parameters

<i>isAlive</i>	Pointer to boolean that will store the connection status.
----------------	-----------------------------------------------------------

4.26.3.13 openVirtualMachineConsole()

```
virStreamPtr VirtualizationFacade::openVirtualMachineConsole (
    const std::string & vmUuid) const [nodiscard]
```

Opens a console stream to a virtual machine.

Parameters

<i>vmUuid</i>	UUID of the virtual machine to connect to.
---------------	--------------------------------------------

Returns

virStreamPtr Pointer to the opened console stream.

4.26.3.14 receiveDataFromConsole()

```
void VirtualizationFacade::receiveDataFromConsole (
    virStreamPtr stream,
    StreamData * streamData) const
```

Receives data from a virtual machine console.

Parameters

<i>stream</i>	Active console stream to read from.
<i>streamData</i>	Pointer to StreamData structure to be filled with received data.

4.26.3.15 sendDataToConsole()

```
void VirtualizationFacade::sendDataToConsole (
    virStreamPtr stream,
    const std::string & data) const
```

Sends data to a virtual machine console.

Parameters

<i>stream</i>	Active console stream to write to.
<i>data</i>	Data string to send to the console.

4.26.3.16 updateVmDevice()

```
void VirtualizationFacade::updateVmDevice (
    const std::string & uuid,
    const std::string & deviceDefinition) const
```

Updates a device configuration in a virtual machine.

Parameters

<i>uuid</i>	UUID of the virtual machine.
<i>deviceDefinition</i>	XML string defining the new device configuration.

The documentation for this class was generated from the following files:

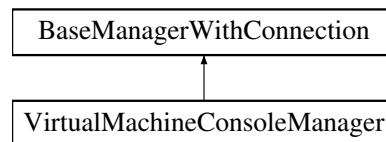
- src/virt/VirtualizationFacade.h
- src/virt/VirtualizationFacade.cpp

4.27 VirtualMachineConsoleManager Class Reference

Manages virtual machine console connections.

```
#include <VirtualMachineConsoleManager.h>
```

Inheritance diagram for VirtualMachineConsoleManager:



Public Member Functions

- virStreamPtr [openVirtualMachineConsole](#) (const std::string &vmUuid) const
Opens a console stream to a virtual machine.
- void [getDataFromStream](#) (virStreamPtr stream, StreamData *streamData) const
Receives data from a virtual machine console stream.
- void [sendDataToStream](#) (virStreamPtr stream, const char *data, int dataSize) const
Sends data to a virtual machine console stream.
- void [closeStream](#) (virStreamPtr stream) const
Closes a console stream and releases resources.
- [BaseManagerWithConnection](#) (ILibvirtWrapper *libvirt)
Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.
- [BaseManagerWithConnection](#) ()
Constructs a [BaseManagerWithConnection](#) with default libvirt wrapper.

Public Member Functions inherited from [BaseManagerWithConnection](#)

- [BaseManagerWithConnection](#) (ILibvirtWrapper *libvirt)
Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.
- [BaseManagerWithConnection](#) ()
Constructs a [BaseManagerWithConnection](#) with default libvirt wrapper.
- void [updateConnection](#) (virConnectPtr conn)
Updates the libvirt connection pointer.

Additional Inherited Members

Protected Member Functions inherited from [BaseManagerWithConnection](#)

- void **checkIfConnectionIsSet () const**
Checks if the connection is set and throws an exception if not.

Protected Attributes inherited from [BaseManagerWithConnection](#)

- [ILibvirtWrapper](#) * **libvirt**
- virConnectPtr **conn**

4.27.1 Detailed Description

Manages virtual machine console connections.

This class provides functionality for opening and managing console streams to virtual machines, allowing for interactive terminal access.

4.27.2 Member Function Documentation

4.27.2.1 [BaseManagerWithConnection\(\)](#)

```
BaseManagerWithConnection::BaseManagerWithConnection (
    ILibvirtWrapper * libvirt) [inline], [explicit]
```

Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.

Parameters

<i>libvirt</i>	Pointer to an ILibvirtWrapper implementation.
----------------	---------------------------------------------------------------

4.27.2.2 [closeStream\(\)](#)

```
void VirtualMachineConsoleManager::closeStream (
    virStreamPtr stream) const
```

Closes a console stream and releases resources.

Parameters

<i>stream</i>	Console stream to close.
---------------	--------------------------

4.27.2.3 [getDataFromStream\(\)](#)

```
void VirtualMachineConsoleManager::getDataFromStream (
    virStreamPtr stream,
    StreamData * streamData) const
```

Receives data from a virtual machine console stream.

Parameters

<i>stream</i>	Active console stream to read from.
<i>streamData</i>	Pointer to StreamData structure to be filled with received data.

4.27.2.4 openVirtualMachineConsole()

```
virStreamPtr VirtualMachineConsoleManager::openVirtualMachineConsole (
    const std::string & vmUuid) const
```

Opens a console stream to a virtual machine.

Parameters

<i>vmUuid</i>	UUID of the virtual machine to connect to.
---------------	--------------------------------------------

Returns

virStreamPtr Pointer to the opened console stream.

4.27.2.5 sendDataToStream()

```
void VirtualMachineConsoleManager::sendDataToStream (
    virStreamPtr stream,
    const char * data,
    int dataSize) const
```

Sends data to a virtual machine console stream.

Parameters

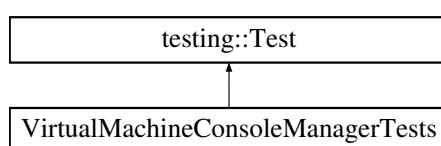
<i>stream</i>	Active console stream to write to.
<i>data</i>	Data to send to the console.
<i>dataSize</i>	Size of data in bytes.

The documentation for this class was generated from the following files:

- src/virt/managers/VirtualMachineConsoleManager.h
- src/virt/managers/VirtualMachineConsoleManager.cpp

4.28 VirtualMachineConsoleManagerTests Class Reference

Inheritance diagram for VirtualMachineConsoleManagerTests:



Protected Attributes

- [LibvirtWrapperMock](#) **mockLibvirt**
- [VirtualMachineConsoleManager](#) **manager**

The documentation for this class was generated from the following file:

- tests/VirtualMachineConsoleManager.cpp

4.29 VirtualMachineInfo Struct Reference

Structure containing basic information about a virtual machine.

```
#include <VirtualMachineInfo.h>
```

Public Attributes

- char **uuid** [128]
The identifier of the virtual machine in libvirt system.
- char **name** [256]
- unsigned long **usedMemory**
- unsigned char **state**

4.29.1 Detailed Description

Structure containing basic information about a virtual machine.

The documentation for this struct was generated from the following file:

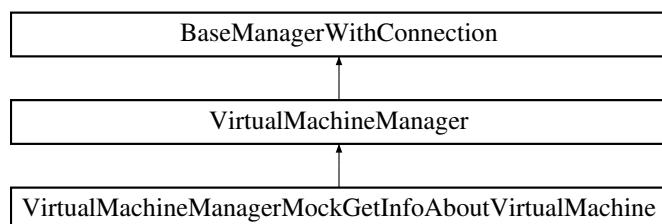
- src/models/VirtualMachineInfo.h

4.30 VirtualMachineManager Class Reference

Manages virtual machines.

```
#include <VirtualMachineManager.h>
```

Inheritance diagram for VirtualMachineManager:



Public Member Functions

- void [createVirtualMachine](#) (const std::string &virtualMachineXml) const
Creates a virtual machine based on an XML configuration.
- virtual [VirtualMachineInfo getInfoAboutVirtualMachine](#) (const std::string &name)
Retrieves information about a virtual machine identified by its name.
- std::vector< [VirtualMachineInfo](#) > [getListOfVirtualMachinesWithInfo](#) ()
Retrieves a list of all virtual machines with their information.
- void [attachDeviceToVirtualMachine](#) (const std::string &uuid, const std::string &deviceDefinition) const
Attaches a device to a virtual machine.
- void [detachDeviceFromVirtualMachine](#) (const std::string &uuid, const std::string &deviceDefinition) const
Detaches a device from a virtual machine.
- void [updateVmDevice](#) (const std::string &uuid, const std::string &deviceDefinition) const
Updates a device configuration in a virtual machine.
- [BaseManagerWithConnection](#) (ILibvirtWrapper *libvirt)
Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.
- [BaseManagerWithConnection](#) ()
Constructs a [BaseManagerWithConnection](#) with default libvirt wrapper.

Public Member Functions inherited from [BaseManagerWithConnection](#)

- [BaseManagerWithConnection](#) (ILibvirtWrapper *libvirt)
Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.
- [BaseManagerWithConnection](#) ()
Constructs a [BaseManagerWithConnection](#) with default libvirt wrapper.
- void [updateConnection](#) (virConnectPtr conn)
Updates the libvirt connection pointer.

Additional Inherited Members

Protected Member Functions inherited from [BaseManagerWithConnection](#)

- void [checkIfConnectionIsSet](#) () const
Checks if the connection is set and throws an exception if not.

Protected Attributes inherited from [BaseManagerWithConnection](#)

- ILibvirtWrapper * **libvirt**
- virConnectPtr **conn**

4.30.1 Detailed Description

Manages virtual machines.

This class handles the connection to the libvirt backend and operations related to virtual machines such as creation and retrieving information.

4.30.2 Member Function Documentation

4.30.2.1 [attachDeviceToVirtualMachine\(\)](#)

```
void VirtualMachineManager::attachDeviceToVirtualMachine (
    const std::string & uuid,
    const std::string & deviceDefinition) const
```

Attaches a device to a virtual machine.

Parameters

<i>uuid</i>	UUID of the virtual machine.
<i>deviceDefinition</i>	XML string defining the device to attach.

4.30.2.2 BaseManagerWithConnection()

```
BaseManagerWithConnection::BaseManagerWithConnection (
    ILibvirtWrapper * libvirt) [inline], [explicit]
```

Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.

Parameters

<i>libvirt</i>	Pointer to an ILibvirtWrapper implementation.
----------------	---------------------------------------------------------------

4.30.2.3 createVirtualMachine()

```
void VirtualMachineManager::createVirtualMachine (
    const std::string & virtualMachineXml) const
```

Creates a virtual machine based on an XML configuration.

Parameters

<i>virtualMachineXml</i>	XML string describing the virtual machine configuration.
--------------------------	----------------------------------------------------------

4.30.2.4 detachDeviceFromVirtualMachine()

```
void VirtualMachineManager::detachDeviceFromVirtualMachine (
    const std::string & uuid,
    const std::string & deviceDefinition) const
```

Detaches a device from a virtual machine.

Parameters

<i>uuid</i>	UUID of the virtual machine.
<i>deviceDefinition</i>	XML string defining the device to detach.

4.30.2.5 getInfoAboutVirtualMachine()

```
VirtualMachineInfo VirtualMachineManager::getInfoAboutVirtualMachine (
    const std::string & name) [virtual]
```

Retrieves information about a virtual machine identified by its name.

Parameters

<code>name</code>	Name string of the virtual machine.
-------------------	-------------------------------------

Returns

`VirtualMachineInfo` Information about the virtual machine.

4.30.2.6 getListOfVirtualMachinesWithInfo()

```
std::vector< VirtualMachineInfo > VirtualMachineManager::getListOfVirtualMachinesWithInfo ()
```

Retrieves a list of all virtual machines with their information.

Parameters

`std::vector<VirtualMachineInfo>` Vector containing information about all VMs.

4.30.2.7 updateVmDevice()

```
void VirtualMachineManager::updateVmDevice (
    const std::string & uuid,
    const std::string & deviceDefinition) const
```

Updates a device configuration in a virtual machine.

Parameters

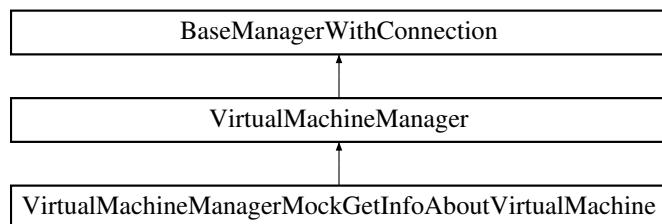
<code>uuid</code>	UUID of the virtual machine.
<code>deviceDefinition</code>	XML string defining the new device configuration.

The documentation for this class was generated from the following files:

- src/virt/managers/VirtualMachineManager.h
- src/virt/managers/VirtualMachineManager.cpp

4.31 VirtualMachineManagerMockGetInfoAboutVirtualMachine Class Reference

Inheritance diagram for VirtualMachineManagerMockGetInfoAboutVirtualMachine:



Public Member Functions

- **VirtualMachineManagerMockGetInfoAboutVirtualMachine** ([ILibvirtWrapper](#) *libvirt)
- **MOCK_METHOD** ([VirtualMachineInfo](#), [getInfoAboutVirtualMachine](#),(const std::string &uuid),(override))

Public Member Functions inherited from [VirtualMachineManager](#)

- void [createVirtualMachine](#) (const std::string &virtualMachineXml) const
Creates a virtual machine based on an XML configuration.
- virtual [VirtualMachineInfo getInfoAboutVirtualMachine](#) (const std::string &name)
Retrieves information about a virtual machine identified by its name.
- std::vector< [VirtualMachineInfo](#) > [getListOfVirtualMachinesWithInfo](#) ()
Retrieves a list of all virtual machines with their information.
- void [attachDeviceToVirtualMachine](#) (const std::string &uuid, const std::string &deviceDefinition) const
Attaches a device to a virtual machine.
- void [detachDeviceFromVirtualMachine](#) (const std::string &uuid, const std::string &deviceDefinition) const
Detaches a device from a virtual machine.
- void [updateVmDevice](#) (const std::string &uuid, const std::string &deviceDefinition) const
Updates a device configuration in a virtual machine.
- [BaseManagerWithConnection](#) ([ILibvirtWrapper](#) *libvirt)
Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.
- **BaseManagerWithConnection** ()
Constructs a [BaseManagerWithConnection](#) with default libvirt wrapper.

Public Member Functions inherited from [BaseManagerWithConnection](#)

- [BaseManagerWithConnection](#) ([ILibvirtWrapper](#) *libvirt)
Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.
- **BaseManagerWithConnection** ()
Constructs a [BaseManagerWithConnection](#) with default libvirt wrapper.
- void [updateConnection](#) (virConnectPtr conn)
Updates the libvirt connection pointer.

Additional Inherited Members

Protected Member Functions inherited from [BaseManagerWithConnection](#)

- void [checkIfConnectionIsSet](#) () const
Checks if the connection is set and throws an exception if not.

Protected Attributes inherited from [BaseManagerWithConnection](#)

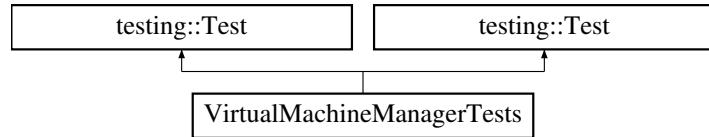
- [ILibvirtWrapper](#) * **libvirt**
- virConnectPtr **conn**

The documentation for this class was generated from the following file:

- tests/mock/VirtualMachineManagerMockGetInfoAboutVirtualMachine.h

4.32 VirtualMachineManagerTests Class Reference

Inheritance diagram for VirtualMachineManagerTests:



Protected Attributes

- `LibvirtWrapperMock` **mockLibvirt**
- `VirtualMachineManager` **manager**
- `VirtualNetworkManager` **manager**

The documentation for this class was generated from the following files:

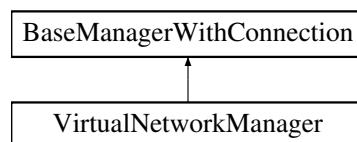
- tests/VirtualMachineManagerTests.cpp
- tests/VirtualNetworkManagerTests.cpp

4.33 VirtualNetworkManager Class Reference

Manages virtual networks.

```
#include <VirtualNetworkManager.h>
```

Inheritance diagram for VirtualNetworkManager:



Public Member Functions

- `virNetworkPtr createNetworkFromXml (const std::string &networkDefinition) const`
Creates a virtual network from an XML definition.
- `void destroyNetwork (const std::string &name) const`
Destroys (stops and removes) a virtual network.
- `std::string getNetworkXmlDefinition (const std::string &name) const`
Retrieves the XML definition of a virtual network.
- `BaseManagerWithConnection (ILibvirtWrapper *libvirt)`
Constructs a `BaseManagerWithConnection` with a specific libvirt wrapper.
- `BaseManagerWithConnection ()`
Constructs a `BaseManagerWithConnection` with default libvirt wrapper.

Public Member Functions inherited from [BaseManagerWithConnection](#)

- **[BaseManagerWithConnection](#) ([ILibvirtWrapper](#) *libvirt)**
Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.
- **[BaseManagerWithConnection](#) ()**
Constructs a [BaseManagerWithConnection](#) with default libvirt wrapper.
- **void [updateConnection](#) (virConnectPtr conn)**
Updates the libvirt connection pointer.

Additional Inherited Members

Protected Member Functions inherited from [BaseManagerWithConnection](#)

- void **[checkIfConnectionIsSet](#) () const**
Checks if the connection is set and throws an exception if not.

Protected Attributes inherited from [BaseManagerWithConnection](#)

- [ILibvirtWrapper](#) * **libvirt**
- virConnectPtr **conn**

4.33.1 Detailed Description

Manages virtual networks.

This class provides functionality for creating, destroying, and managing virtual networks using libvirt.

4.33.2 Member Function Documentation

4.33.2.1 [BaseManagerWithConnection\(\)](#)

```
BaseManagerWithConnection::BaseManagerWithConnection (
    ILibvirtWrapper * libvirt) [inline], [explicit]
```

Constructs a [BaseManagerWithConnection](#) with a specific libvirt wrapper.

Parameters

<i>libvirt</i>	Pointer to an ILibvirtWrapper implementation.
----------------	---------------------------------------------------------------

4.33.2.2 [createNetworkFromXml\(\)](#)

```
virNetworkPtr VirtualNetworkManager::createNetworkFromXml (
    const std::string & networkDefinition) const
```

Creates a virtual network from an XML definition.

Parameters

<i>networkDefinition</i>	XML string defining the network configuration.
--------------------------	------------------------------------------------

Returns

`virNetworkPtr` Pointer to the created network.

4.33.2.3 `destroyNetwork()`

```
void VirtualNetworkManager::destroyNetwork (
    const std::string & name) const
```

Destroys (stops and removes) a virtual network.

Parameters

<i>name</i>	Name of the network to destroy.
-------------	---------------------------------

4.33.2.4 `getNetworkXmlDefinition()`

```
std::string VirtualNetworkManager::getNetworkXmlDefinition (
    const std::string & name) const [nodiscard]
```

Retrieves the XML definition of a virtual network.

Parameters

<i>name</i>	Name of the network.
-------------	----------------------

Returns

`std::string` XML definition of the network.

The documentation for this class was generated from the following files:

- `src/virt/managers/VirtualNetworkManager.h`
- `src/virt/managers/VirtualNetworkManager.cpp`

Chapter 5

File Documentation

5.1 PacketSnifferException.h

```
00001 #ifndef SNIFFEREXCEPTION_H
00002 #define SNIFFEREXCEPTION_H
00003 #include <string>
00004 #include <utility>
00005
00011 class PacketSnifferException final : public std::exception
00012 {
00013     std::string message;
00014
00015 public:
00021     explicit PacketSnifferException(std::string msg) : message(std::move(msg))
00022     {
00023     }
00024
00030     [[nodiscard]] const char* what() const noexcept override
00031     {
00032         return message.c_str();
00033     }
00034 };
00035 #endif //SNIFFEREXCEPTION_H
```

5.2 VirtualizationException.h

```
00001 #ifndef VIRTUALIZATIONEXCEPTION_H
00002 #define VIRTUALIZATIONEXCEPTION_H
00003 #include <exception>
00004 #include <string>
00005 #include <utility>
00006
00012 class VirtualizationException final : public std::exception {
00013     std::string message;
00014
00015 public:
00021     explicit VirtualizationException(std::string msg) : message(std::move(msg)) {
00022     }
00023
00029     [[nodiscard]] const char *what() const noexcept override {
00030         return message.c_str();
00031     }
00032 };
00033 #endif //VIRTUALIZATIONEXCEPTION_H
```

5.3 ILibpcapWrapper.h

```
00001 #ifndef ILIBPCAPWRAPPER_H
00002 #define ILIBPCAPWRAPPER_H
00003 #include <string>
00004 #include <pcap/pcap.h>
```

```

00005
00012 class ILibpcapWrapper
00013 {
00014 public:
00015     virtual ~ILibpcapWrapper() = default;
00016
00024     virtual pcap_t* openHandlerLive(const std::string& interfaceName, char* errBuff) = 0;
00025
00031     virtual int getLinkLayerType(pcap_t* handler) = 0;
00032
00038     virtual void closeHandler(pcap_t* handler) = 0;
00039
00047     virtual int listenForPackets(pcap_t* handler, pcap_handler callback, u_char* args) = 0;
00048
00054     virtual void close(pcap_t* handler) = 0;
00055 };
00056
00057 #endif //ILIBPCAPWRAPPER_H

```

5.4 ILibvirtWrapper.h

```

00001 #ifndef LIBVIRTWRAPPER_H
00002 #define LIBVIRTWRAPPER_H
00003 #include <string>
00004 #include <libvirt/libvirt.h>
00005
00012 class ILibvirtWrapper
00013 {
00014 public:
00015     virtual ~ILibvirtWrapper() = default;
00016
00023     virtual virConnectPtr connectOpen(const char* name) = 0;
00024
00032     virtual virDomainPtr createVirtualMachineFromXml(virConnectPtr conn, const char* xmlConfig) = 0;
00033
00040     virtual void getUuidFromDomain(virDomainPtr domain, char* uuid) = 0;
00041
00049     virtual int getNodeInfo(virConnectPtr conn, virNodeInfoPtr info) = 0;
00050
00058     virtual int getLibVersion(virConnectPtr conn, unsigned long* version) = 0;
00059
00067     virtual int getDriverVersion(virConnectPtr conn, unsigned long* version) = 0;
00068
00075     virtual std::string getConnectUrl(virConnectPtr conn) = 0;
00076
00083     virtual std::string getDriverType(virConnectPtr conn) = 0;
00084
00090     virtual std::string getLastErrorMessage() = 0;
00091
00099     virtual virDomainPtr domainLookupByName(virConnectPtr conn, std::string name) = 0;
00100
00108     virtual int domainGetInfo(virDomainPtr domain, virDomainInfo& domainInfo) = 0;
00109
00117     virtual int getDomainUUID(virDomainPtr domain, std::string& uuid) = 0;
00118
00126     virtual int getListOfAllDomains(virConnectPtr conn, virDomainPtr** domains) = 0;
00127
00134     virtual std::string getDomainName(virDomainPtr domain) = 0;
00135
00141     virtual void freeDomain(virDomainPtr domain) = 0;
00142
00149     virtual int connectionIsAlive(virConnectPtr conn) = 0;
00150
00157     virtual virStreamPtr createNewStream(virConnectPtr conn) = 0;
00158
00166     virtual int openDomainConsole(virDomainPtr domain, virStreamPtr stream) = 0;
00167
00175     virtual virDomainPtr domainLookupByUuid(virConnectPtr conn, const std::string& uuid) = 0;
00176
00185     virtual int receiveDataFromStream(virStreamPtr stream, char* buffer, int bufferSize) = 0;
00186
00194     virtual void sendDataToStream(virStreamPtr stream, const char* buffer, int bufferSize) = 0;
00195
00201     virtual void finishAndFreeStream(virStreamPtr stream) = 0;
00202
00210     virtual virNetworkPtr createNetworkFromXml(virConnectPtr conn, const std::string&
00211         networkDefinition) = 0;
00211
00219     virtual int attachDeviceToVm(virDomainPtr domain, const std::string& deviceDefinition) = 0;
00220
00228     virtual int detachDeviceFromVm(virDomainPtr domain, const std::string& deviceDefinition) = 0;
00229
00237     virtual int updateVmDevice(virDomainPtr domain, const std::string& deviceDefinition) = 0;

```

```

00238
00246     virtual virNetworkPtr getNetworkByName(virConnectPtr conn, const std::string& name) = 0;
00247
00254     virtual int destroyNetwork(virNetworkPtr network) = 0;
00255
00262     virtual std::string getNetworkDefinition(virNetworkPtr network) = 0;
00263 };
00264
00265
00266 #endif // LIBVIRTWRAPPER_H

```

5.5 ConnectionInfo.h

```

00001 #ifndef CONNECTIONINFO_H
00002 #define CONNECTIONINFO_H
00003 #include "Version.h"
00004
00011 struct ConnectionInfo {
00015     unsigned int cpuCount;
00016
00020     unsigned int cpuFreq;
00021
00025     unsigned long totalMemory;
00026
00030     char connectionUrl[256];
00031
00035     char driverType[256];
00036
00040     Version libVersion;
00041
00045     Version driverVersion;
00046 };
00047
00048 #endif //CONNECTIONINFO_H

```

5.6 ExecutionInfo.h

```

00001 #ifndef EXECUTIONINFO_H
00002 #define EXECUTIONINFO_H
00003
00009 struct ExecutionInfo
00010 {
00014     bool errorOccurred;
00015
00019     char msg[128];
00020 };
00021
00022 #endif // EXECUTIONINFO_H

```

5.7 ListenCallbackArgs.h

```

00001 #ifndef LISTENCALLBACKARGS_H
00002 #define LISTENCALLBACKARGS_H
00003 #include "../packetsniffer/PacketSniffer.h"
00004
00011 struct ListenCallbackArgs
00012 {
00016     PacketSniffer* sniffer;
00017
00021     std::string interfaceName;
00022 };
00023
00024 #endif //LISTENCALLBACKARGS_H

```

5.8 NetworkDefinition.h

```

00001 #ifndef NETWORKDEFINITION_H
00002 #define NETWORKDEFINITION_H
00003

```

```
00009 struct NetworkDefinition
00010 {
00014     char content[4096];
00015 };
00016
00017 #endif
```

5.9 Packet.h

```
00001 #ifndef PACKET_H
00002 #define PACKET_H
00003
00010 struct Packet
00011 {
00015     char interfaceName[64];
00016
00020     unsigned char* content;
00021
00025     int contentLength;
00026
00030     unsigned int timestampMicroseconds;
00031 };
00032
00033 #endif //PACKET_H
```

5.10 StreamData.h

```
00001 #ifndef STREAMRESPONSE_H
00002 #define STREAMRESPONSE_H
00003
00009 struct StreamData
00010 {
00014     char buffer[255];
00015
00019     bool isStreamBroken;
00020 };
00021
00022 #endif // STREAMRESPONSE_H
```

5.11 Version.h

```
00001 #ifndef VERSION_H
00002 #define VERSION_H
00003
00009 struct Version {
00013     unsigned int major;
00014
00018     unsigned int minor;
00019
00023     unsigned int patch;
00024 };
00025
00026 #endif //VERSION_H
```

5.12 VirtualMachineInfo.h

```
00001 #ifndef VIRTUALMACHINEINFO_H
00002 #define VIRTUALMACHINEINFO_H
00003
00007 struct VirtualMachineInfo
00008 {
00012     char uid[128];
00013     char name[256];
00014     unsigned long usedMemory;
00015     unsigned char state;
00016 };
00017
00018 #endif //VIRTUALMACHINEINFO_H
```

5.13 PacketSniffer.h

```

00001 #ifndef PACKETSNIFFER_H
00002 #define PACKETSNIFFER_H
00003 #include <queue>
00004 #include <string>
00005
00006 #include "../interfaces/ILibpcapWrapper.h"
00007 #include "../models/Packet.h"
00008 #include "../wrappers/LibpcapWrapper.h"
00009
00010 class PacketSniffer
00011 {
00012     ILibpcapWrapper* pcap;
00013     char errBuffer[PCAP_ERRBUF_SIZE]{};
00014     std::queue<Packet> packetsQueue;
00015
00016 public:
00017     PacketSniffer()
00018     {
00019         pcap = new LibpcapWrapper();
00020     }
00021
00022     [[nodiscard]] pcap_t* openSnifferHandler(const std::string& interfaceName);
00023     [[nodiscard]] bool listenForPacket(pcap_t* snifferHandler, const std::string& interfaceName);
00024     void closeAndStopListening(pcap_t* handler) const;
00025
00026     [[nodiscard]] Packet getPacketFromQueue();
00027     [[nodiscard]] int getNumberOfReceivedPackets() const;
00028
00029 private:
00030     static void handlePackets(u_char* args, const pcap_pkthdr* header, const u_char* packet);
00031 };
00032
00033 #endif //PACKETSNIFFER_H

```

5.14 ExecutionInfoObtainer.h

```

00001 #ifndef EXECUTIONINFOOBTAINER_H
00002 #define EXECUTIONINFOOBTAINER_H
00003 #include <functional>
00004
00005 #include "../models/ExecutionInfo.h"
00006
00007 class ExecutionInfoObtainer
00008 {
00009 public:
00010     static void runAndObtainExecutionInfo(ExecutionInfo* executionInfo, const std::function<void()>& func);
00011 };
00012
00013 #endif // EXECUTIONINFOOBTAINER_H

```

5.15 StringUtils.h

```

00001 #ifndef STRINGTOCHARUTILS_H
00002 #define STRINGTOCHARUTILS_H
00003 #include <string>
00004
00005 class StringUtils
00006 {
00007 public:
00008     static void copyStringToCharArray(const std::string& src, char* charArray, int length);
00009 };
00010
00011
00012 #endif //STRINGTOCHARUTILS_H

```

5.16 VersionUtils.h

```
00001 #ifndef VERSIONUTILS_H
```

```

00002 #define VERSIONUTILS_H
00003 #include "../models/Version.h"
00004
00010 class VersionUtils {
00011 public:
00021     static Version getVersion(unsigned long version);
00022 };
00023
00024 #endif //VERSIONUTILS_H

```

5.17 BaseManagerWithConnection.h

```

00001 #ifndef BASEMANAGERWITHCONNECTION_H
00002 #define BASEMANAGERWITHCONNECTION_H
00003 #include "../../wrappers/LibvirtWrapper.h"
00004 #include "../../interfaces/ILibvirtWrapper.h"
00005
00013 class BaseManagerWithConnection
00014 {
00015 protected:
00016     ILibvirtWrapper* libvirt;
00017     virConnectPtr conn;
00018
00019 public:
00020     virtual ~BaseManagerWithConnection() = default;
00021
00027     explicit BaseManagerWithConnection(ILibvirtWrapper* libvirt)
00028         : libvirt(libvirt), conn(nullptr)
00029     {
00030     }
00031
00035     explicit BaseManagerWithConnection(): conn(nullptr)
00036     {
00037         libvirt = new LibvirtWrapper();
00038     }
00039
00045     void updateConnection(virConnectPtr conn);
00046
00047 protected:
00051     void checkIfConnectionIsSet() const;
00052 };
00053
00054 #endif //BASEMANAGERWITHCONNECTION_H

```

5.18 ConnectionManager.h

```

00001 #ifndef CONNECTIONMANAGER_H
00002 #define CONNECTIONMANAGER_H
00003 #include <optional>
00004 #include <string>
00005
00006 #include "../../wrappers/LibvirtWrapper.h"
00007 #include "../../models/ConnectionInfo.h"
00008
00015 class ConnectionManager
00016 {
00017     ILibvirtWrapper* libvirt;
00018     virConnectPtr conn = nullptr;
00019
00020 public:
00021     virtual ~ConnectionManager() = default;
00022
00028     explicit ConnectionManager(ILibvirtWrapper* libvirt)
00029         : libvirt(libvirt)
00030     {
00031     }
00032
00036     explicit ConnectionManager()
00037     {
00038         libvirt = new LibvirtWrapper();
00039     }
00040
00047     void initializeConnection(const std::optional<std::string>& customConnectionStringUrl = std::nullopt);
00048
00054     ConnectionInfo getConnectionInfo() const;
00055
00061     bool isConnectionAlive() const;
00062

```

```

00068     virConnectPtr getConnection() const;
00069 };
00070
00071
00072 #endif //CONNECTIONMANAGER_H

```

5.19 VirtualMachineConsoleManager.h

```

00001 #ifndef VIRTUALMACHINECONSOLEMANAGER_H
00002 #define VIRTUALMACHINECONSOLEMANAGER_H
00003 #include "BaseManagerWithConnection.h"
00004 #include "../../wrappers/LibvirtWrapper.h"
00005 #include "../../interfaces/ILibvirtWrapper.h"
00006 #include "../../models/StreamData.h"
00007
00014 class VirtualMachineConsoleManager : public BaseManagerWithConnection
00015 {
00016 public:
00017     using BaseManagerWithConnection::BaseManagerWithConnection;
00018
00025     virStreamPtr openVirtualMachineConsole(const std::string& vmUuid) const;
00026
00033     void getDataFromStream(virStreamPtr stream, StreamData* streamData) const;
00034
00042     void sendDataToStream(virStreamPtr stream, const char* data, int dataSize) const;
00043
00049     void closeStream(virStreamPtr stream) const;
00050 };
00051
00052
00053 #endif //VIRTUALMACHINECONSOLEMANAGER_H

```

5.20 VirtualMachineManager.h

```

00001 #ifndef VIRTUALMACHINEMANAGER_H
00002 #define VIRTUALMACHINEMANAGER_H
00003 #include <string>
00004 #include <vector>
00005 #include <libvirt/libvirt.h>
00006
00007 #include "BaseManagerWithConnection.h"
00008 #include "../../models/VirtualMachineInfo.h"
00009
00017 class VirtualMachineManager : public BaseManagerWithConnection
00018 {
00019 public:
00020     using BaseManagerWithConnection::BaseManagerWithConnection;
00021
00027     void createVirtualMachine(const std::string& virtualMachineXml) const;
00028
00035     virtual VirtualMachineInfo getInfoAboutVirtualMachine(const std::string& name);
00036
00042     std::vector<VirtualMachineInfo> getListOfVirtualMachinesWithInfo();
00043
00050     void attachDeviceToVirtualMachine(const std::string& uuid, const std::string& deviceDefinition)
00051         const;
00058     void detachDeviceFromVirtualMachine(const std::string& uuid, const std::string& deviceDefinition)
00059         const;
00066     void updateVmDevice(const std::string& uuid, const std::string& deviceDefinition) const;
00067
00068 private:
00075     [[nodiscard]] virDomainPtr getVirtualMachineByName(const std::string& name) const;
00076
00083     [[nodiscard]] virDomainPtr getVirtualMachineByUuid(const std::string& uuid) const;
00084 };
00085
00086 #endif //VIRTUALMACHINEMANAGER_H

```

5.21 VirtualNetworkManager.h

```
00001 #ifndef VIRTUALNETWORKMANAGER_H
```

```

00002 #define VIRTUALNETWORKMANAGER_H
00003 #include "BaseManagerWithConnection.h"
00004 #include "../../wrappers/LibvirtWrapper.h"
00005 #include "../../../../interfaces/ILibvirtWrapper.h"
00006
00013 class VirtualNetworkManager : public BaseManagerWithConnection
00014 {
00015 public:
00016     using BaseManagerWithConnection::BaseManagerWithConnection;
00017
00024     virNetworkPtr createNetworkFromXml(const std::string& networkDefinition) const;
00025
00031     void destroyNetwork(const std::string& name) const;
00032
00039     [[nodiscard]] std::string getNetworkXmlDefinition(const std::string& name) const;
00040 };
00041
00042 #endif //VIRTUALNETWORKMANAGER_H

```

5.22 VirtualizationFacade.h

```

00001 #ifndef VIRTUALIZATIONFACADE_H
00002 #define VIRTUALIZATIONFACADE_H
00003 #include "managers/ConnectionManager.h"
00004 #include "managers/VirtualMachineConsoleManager.h"
00005 #include "managers/VirtualMachineManager.h"
00006 #include "../models/StreamData.h"
00007 #include "managers/VirtualNetworkManager.h"
00008
00016 class VirtualizationFacade
00017 {
00018     ConnectionManager* connManager = nullptr;
00019     VirtualMachineManager* vmManager = nullptr;
00020     VirtualMachineConsoleManager* vmConsoleManager = nullptr;
00021     VirtualNetworkManager* networkManager = nullptr;
00022
00023 public:
00024     virtual ~VirtualizationFacade() = default;
00025
00031     explicit VirtualizationFacade(ILibvirtWrapper* libvirt);
00032
00036     explicit VirtualizationFacade();
00037
00043     void initializeConnection(const char* customConnectionUrl) const;
00044
00050     void getConnectionInfo(ConnectionInfo* infoPtr) const;
00051
00057     void createVirtualMachine(const std::string& virtualMachineXml) const;
00058
00065     void getInfoAboutVirtualMachine(VirtualMachineInfo* virtualMachineInfo, const std::string& name) const;
00066
00073     void getListOfVirtualMachinesWithInfo(VirtualMachineInfo** arrayOfVirtualMachines,
00074                                         int* numberofVirtualMachines) const;
00075
00081     void isConnectionAlive(bool* isAlive) const;
00082
00089     [[nodiscard]] virStreamPtr openVirtualMachineConsole(const std::string& vmUuid) const;
00090
00097     void receiveDataFromConsole(virStreamPtr stream, StreamData* streamData) const;
00098
00105     void sendDataToConsole(virStreamPtr stream, const std::string& data) const;
00106
00112     void closeStream(virStreamPtr stream) const;
00113
00120     [[nodiscard]] virNetworkPtr createVirtualNetworkFromXml(const std::string& networkDefinition) const;
00121
00128     void attachDeviceToVm(const std::string& uuid, const std::string& deviceDefinition) const;
00129
00136     void detachDeviceFromVm(const std::string& uuid, const std::string& deviceDefinition) const;
00137
00144     void updateVmDevice(const std::string& uuid, const std::string& deviceDefinition) const;
00145
00151     void destroyNetwork(const std::string& name) const;
00152
00159     [[nodiscard]] std::string getNetworkDefinition(const std::string& name) const;
00160 };
00161
00162 #endif //VIRTUALIZATIONFACADE_H

```

5.23 LibpcapWrapper.h

```

00001 #ifndef LIBPCAPWRAPPER_H
00002 #define LIBPCAPWRAPPER_H
00003 #include "../interfaces/ILibpcapWrapper.h"
00004
00011 class LibpcapWrapper final : public ILibpcapWrapper
00012 {
00013 public:
00021     pcap_t* openHandlerLive(const std::string& interfaceName, char* errBuff) override;
00022
00029     int getLinkLayerType(pcap_t* handler) override;
00030
00036     void closeHandler(pcap_t* handler) override;
00037
00046     int listenForPackets(pcap_t* handler, pcap_handler callback, u_char* args) override;
00047
00053     void close(pcap_t* handler) override;
00054 };
00055
00056 #endif //LIBPCAPWRAPPER_H

```

5.24 LibvirtWrapper.h

```

00001 #ifndef LIBVIRTWRAPPER_H
00002 #define LIBVIRTWRAPPER_H
00003 #include "../interfaces/ILibvirtWrapper.h"
00004 #include <libvirt/virterror.h>
00005
00011 class LibvirtWrapper final : public ILibvirtWrapper
00012 {
00013 public:
00020     virConnectPtr connectOpen(const char* connectionUri) override;
00021
00029     virDomainPtr createVirtualMachineFromXml(virConnectPtr conn, const char* xmlConfig) override;
00030
00037     void getUuidFromDomain(virDomainPtr domain, char* uuid) override;
00038
00046     int getNodeInfo(virConnectPtr conn, virNodeInfoPtr info) override;
00047
00055     int getLibVersion(virConnectPtr conn, unsigned long* libVersion) override;
00056
00064     int getDriverVersion(virConnectPtr conn, unsigned long* version) override;
00065
00072     std::string getConnectUrl(virConnectPtr conn) override;
00073
00080     std::string getDriverType(virConnectPtr conn) override;
00081
00087     std::string getLastErrorMessage() override;
00088
00096     virDomainPtr domainLookupByName(virConnectPtr conn, std::string name) override;
00097
00105     int domainGetInfo(virDomainPtr domain, virDomainInfo& domainInfo) override;
00106
00114     int getDomainUUID(virDomainPtr domain, std::string& uuid) override;
00115
00123     int getListOfAllDomains(virConnectPtr conn, virDomainPtr** domains) override;
00124
00131     std::string getDomainName(virDomainPtr domain) override;
00132
00138     void freeDomain(virDomainPtr domain) override;
00139
00146     int connectionIsAlive(virConnectPtr conn) override;
00147
00154     virStreamPtr createNewStream(virConnectPtr conn) override;
00155
00163     int openDomainConsole(virDomainPtr domain, virStreamPtr stream) override;
00164
00172     virDomainPtr domainLookupByUuid(virConnectPtr conn, const std::string& uuid) override;
00173
00182     int receiveDataFromStream(virStreamPtr stream, char* buffer, int bufferSize) override;
00183
00191     void sendDataToStream(virStreamPtr stream, const char* buffer, int bufferSize) override;
00192
00198     void finishAndFreeStream(virStreamPtr stream) override;
00199
00207     virNetworkPtr createNetworkFromXml(virConnectPtr conn, const std::string& networkDefinition)
00208         override;
00216     int attachDeviceToVm(virDomainPtr domain, const std::string& deviceDefinition) override;
00217
00225     int detachDeviceFromVm(virDomainPtr domain, const std::string& deviceDefinition) override;

```

```

00226
00227     int updateVmDevice(virDomainPtr domain, const std::string& deviceDefinition) override;
00228
00229     virNetworkPtr getNetworkByName(virConnectPtr conn, const std::string& name) override;
00230
00231     int destroyNetwork(virNetworkPtr network) override;
00232
00233     std::string getNetworkDefinition(virNetworkPtr network) override;
00234
00235 };
00236
00237
00238
00239 #endif //LIBVIRTWRAPPER_H

```

5.25 LibvirtWrapperMock.h

```

00001 #ifndef LIBVIRTWRAPPERMOCK_H
00002 #define LIBVIRTWRAPPERMOCK_H
00003 #include <gmock/gmock.h>
00004 #include "interfaces/ILibvirtWrapper.h"
00005
00006 class LibvirtWrapperMock final : public ILibvirtWrapper
00007 {
00008 public:
00009     MOCK_METHOD(virConnectPtr, connectOpen, (const char *name), (override));
00010     MOCK_METHOD(virDomainPtr, createVirtualMachineFromXml, (virConnectPtr conn, const char
00011 *xmlConfig), (override));
00012     MOCK_METHOD(void, getUuidFromDomain, (virDomainPtr domain, char *uuid), (override));
00013     MOCK_METHOD(int, getNodeInfo, (virConnectPtr conn, virNodeInfoPtr info), (override));
00014     MOCK_METHOD(int, getLibVersion, (virConnectPtr conn, unsigned long *version), (override));
00015     MOCK_METHOD(int, getDriverVersion, (virConnectPtr conn, unsigned long *version), (override));
00016     MOCK_METHOD(std::string, getConnectUrl, (virConnectPtr conn), (override));
00017     MOCK_METHOD(std::string, getDriverType, (virConnectPtr conn), (override));
00018     MOCK_METHOD(std::string, getLastErrorMessage, (), (override));
00019     MOCK_METHOD(virDomainPtr, domainLookupByName, (virConnectPtr conn, std::string name), (override));
00020     MOCK_METHOD(int, domainGetInfo, (virDomainPtr domain, virDomainInfo& domainInfo), (override));
00021     MOCK_METHOD(int, getDomainUUID, (virDomainPtr domain, std::string& uuid), (override));
00022     MOCK_METHOD(int, getListOfAllDomains, (virConnectPtr conn, virDomainPtr **domains), (override));
00023     MOCK_METHOD(std::string, getDomainName, (virDomainPtr domain), (override));
00024     MOCK_METHOD(void, freeDomain, (virDomainPtr domain), (override));
00025     MOCK_METHOD(int, connectionIsAlive, (virConnectPtr conn), (override));
00026     MOCK_METHOD(virStreamPtr, createNewStream, (virConnectPtr conn), (override));
00027     MOCK_METHOD(int, openDomainConsole, (virDomainPtr domain, virStreamPtr stream), (override));
00028     MOCK_METHOD(virDomainPtr, domainLookupByUuid, (virConnectPtr conn, const std::string& uuid),
00029 (override));
00030     MOCK_METHOD(int, receiveDataFromStream, (virStreamPtr stream, char* buffer, int bufferSize),
00031 (override));
00032     MOCK_METHOD(void, sendDataToStream, (virStreamPtr stream, const char* buffer, int bufferSize),
00033 (override));
00034     MOCK_METHOD(void, finishAndFreeStream, (virStreamPtr stream), (override));
00035     MOCK_METHOD(virNetworkPtr, createNetworkFromXml, (virConnectPtr conn, const std::string&
00036 networkDefinition),
00037 (override));
00038     MOCK_METHOD(int, attachDeviceToVm, (virDomainPtr domain, const std::string& deviceDefinition),
00039 (override));
00040     MOCK_METHOD(int, detachDeviceFromVm, (virDomainPtr domain, const std::string& deviceDefinition),
00041 (override));
00042     MOCK_METHOD(int, updateVmDevice, (virDomainPtr domain, const std::string& deviceDefinition),
00043 (override));
00044     MOCK_METHOD(virNetworkPtr, getNetworkByName, (virConnectPtr conn, const std::string& name),
00045 (override));
00046     MOCK_METHOD(int, destroyNetwork, (virNetworkPtr network), (override));
00047     MOCK_METHOD(std::string, getNetworkDefinition, (virNetworkPtr network), (override));
00048 };
00049
00050
00051
00052 #endif //LIBVIRTWRAPPERMOCK_H

```

5.26 VirtualMachineManagerMockGetInfoAboutVirtualMachine.h

```

00001 #ifndef VIRTUALMACHINEMANAGERMOCKGETINFOABOUTVIRTUALMACHINE_H
00002 #define VIRTUALMACHINEMANAGERMOCKGETINFOABOUTVIRTUALMACHINE_H
00003 #include <gmock/gmock-function-mocker.h>
00004
00005 #include "virt/managers/VirtualMachineManager.h"
00006
00007 class VirtualMachineManagerMockGetInfoAboutVirtualMachine final : public VirtualMachineManager
00008 {
00009 public:

```

```
00010     explicit VirtualMachineManagerMockGetInfoAboutVirtualMachine(ILibvirtWrapper* libvirt)
00011         : VirtualMachineManager(libvirt)
00012     {
00013     }
00014
00015     MOCK_METHOD(VirtualMachineInfo, getInfoAboutVirtualMachine, (const std::string &uuid),
00016     (override));
00017 }
00018 #endif //VIRTUALMACHINEMANAGERMOCKGETINFOABOUTVIRTUALMACHINE_H
```

5.27 TestingUtils.h

```
00001 #ifndef TESTINGUTILS_H
00002 #define TESTINGUTILS_H
00003 #include <functional>
00004 #include <string>
00005
00006
00007 class TestingUtils
00008 {
00009 public:
00010     static void expectThrowWithMessage(const std::function<void()>& func, const std::string&
00011     expectedMessage);
00012 };
00013
00014 #endif //TESTINGUTILS_H
```


Index

attachDeviceToVirtualMachine
 VirtualMachineManager, 63

attachDeviceToVm
 ILibvirtWrapper, 17
 LibvirtWrapper, 30
 VirtualizationFacade, 55

BaseManagerWithConnection, 7
 BaseManagerWithConnection, 8
 updateConnection, 8
 VirtualMachineConsoleManager, 60
 VirtualMachineManager, 64
 VirtualNetworkManager, 68

close
 ILibpcapWrapper, 14
 LibpcapWrapper, 27

closeAndStopListening
 PacketSniffer, 45

closeHandler
 ILibpcapWrapper, 14
 LibpcapWrapper, 27

closeStream
 VirtualizationFacade, 55
 VirtualMachineConsoleManager, 60

ConnectionInfo, 8

connectionIsAlive
 ILibvirtWrapper, 17
 LibvirtWrapper, 30

ConnectionManager, 9
 ConnectionManager, 10
 getConnection, 10
 getConnectionInfo, 10
 initializeConnection, 10
 isConnectionAlive, 11

ConnectionManagerTests, 11

connectOpen
 ILibvirtWrapper, 17
 LibvirtWrapper, 31

copyStringToCharArray
 StringUtils, 48

createNetworkFromXml
 ILibvirtWrapper, 18
 LibvirtWrapper, 31
 VirtualNetworkManager, 68

createNewStream
 ILibvirtWrapper, 18
 LibvirtWrapper, 31

createVirtualMachine
 VirtualizationFacade, 55

VirtualMachineManager, 64

createVirtualMachineFromXml
 ILibvirtWrapper, 18
 LibvirtWrapper, 32

createVirtualNetworkFromXml
 VirtualizationFacade, 55

destroyNetwork
 ILibvirtWrapper, 19
 LibvirtWrapper, 32
 VirtualizationFacade, 56
 VirtualNetworkManager, 69

detachDeviceFromVirtualMachine
 VirtualMachineManager, 64

detachDeviceFromVm
 ILibvirtWrapper, 19
 LibvirtWrapper, 32
 VirtualizationFacade, 56

domainGetInfo
 ILibvirtWrapper, 19
 LibvirtWrapper, 33

domainLookupByName
 ILibvirtWrapper, 20
 LibvirtWrapper, 33

domainLookupByUuid
 ILibvirtWrapper, 20
 LibvirtWrapper, 33

ExecutionInfo, 11

ExecutionInfoObtainer, 12
 runAndObtainExecutionInfo, 12

ExecutionInfoObtainerTests, 13

finishAndFreeStream
 ILibvirtWrapper, 20
 LibvirtWrapper, 34

freeDomain
 ILibvirtWrapper, 21
 LibvirtWrapper, 34

getConnection
 ConnectionManager, 10

getConnectionInfo
 ConnectionManager, 10
 VirtualizationFacade, 56

getConnectUrl
 ILibvirtWrapper, 21
 LibvirtWrapper, 34

getDataFromStream
 VirtualMachineConsoleManager, 60

getDomainName
ILibvirtWrapper, 21
LibvirtWrapper, 34
getDomainUUID
ILibvirtWrapper, 21
LibvirtWrapper, 36
getDriverType
ILibvirtWrapper, 22
LibvirtWrapper, 36
getDriverVersion
ILibvirtWrapper, 22
LibvirtWrapper, 36
getInfoAboutVirtualMachine
VirtualizationFacade, 56
VirtualMachineManager, 64
getLastErrorCode
ILibvirtWrapper, 22
LibvirtWrapper, 37
getLibVersion
ILibvirtWrapper, 23
LibvirtWrapper, 37
getLinkLayerType
ILibpcapWrapper, 14
LibpcapWrapper, 27
getListAllDomains
ILibvirtWrapper, 23
LibvirtWrapper, 37
getListVirtualMachinesWithInfo
VirtualizationFacade, 57
VirtualMachineManager, 65
getNetworkByName
ILibvirtWrapper, 23
LibvirtWrapper, 38
getNetworkDefinition
ILibvirtWrapper, 24
LibvirtWrapper, 38
VirtualizationFacade, 57
getNetworkXmlDefinition
VirtualNetworkManager, 69
getNodeInfo
ILibvirtWrapper, 24
LibvirtWrapper, 38
getNumberOfReceivedPackets
PacketSniffer, 45
getPacketFromQueue
PacketSniffer, 45
getUuidFromDomain
ILibvirtWrapper, 24
LibvirtWrapper, 39
getVersion
VersionUtils, 50

ILibpcapWrapper, 13
close, 14
closeHandler, 14
getLinkLayerType, 14
listenForPackets, 14
openHandlerLive, 15
ILibvirtWrapper, 15

attachDeviceToVm, 17
connectionIsAlive, 17
connectOpen, 17
createNetworkFromXml, 18
createNewStream, 18
createVirtualMachineFromXml, 18
destroyNetwork, 19
detachDeviceFromVm, 19
domainGetInfo, 19
domainLookupByName, 20
domainLookupByUuid, 20
finishAndFreeStream, 20
freeDomain, 21
getConnectUrl, 21
getDomainName, 21
getDomainUUID, 21
getDriverType, 22
getDriverVersion, 22
getLastErrorCode, 22
getLibVersion, 23
getListAllDomains, 23
getNetworkByName, 23
getNetworkDefinition, 24
getNodeInfo, 24
getUuidFromDomain, 24
openDomainConsole, 25
receiveDataFromStream, 25
sendDataToStream, 25
updateVmDevice, 26

initializeConnection
ConnectionManager, 10
VirtualizationFacade, 57
isConnectionAlive
ConnectionManager, 11
VirtualizationFacade, 57

LibpcapWrapper, 26
close, 27
closeHandler, 27
getLinkLayerType, 27
listenForPackets, 28
openHandlerLive, 28

LibvirtWrapper, 29
attachDeviceToVm, 30
connectionIsAlive, 30
connectOpen, 31
createNetworkFromXml, 31
createNewStream, 31
createVirtualMachineFromXml, 32
destroyNetwork, 32
detachDeviceFromVm, 32
domainGetInfo, 33
domainLookupByName, 33
domainLookupByUuid, 33
finishAndFreeStream, 34
freeDomain, 34
getConnectUrl, 34
getDomainName, 34
getDomainUUID, 36

getDriverType, 36
getDriverVersion, 36
getLastError, 37
getLibVersion, 37
getListAllDomains, 37
getNetworkByName, 38
getNetworkDefinition, 38
getNodeInfo, 38
getUuidFromDomain, 39
openDomainConsole, 39
receiveDataFromStream, 39
sendDataToStream, 40
updateVmDevice, 40
LibvirtWrapperMock, 41
ListenCallbackArgs, 43
listenForPacket
 PacketSniffer, 46
listenForPackets
 ILibpcapWrapper, 14
 LibpcapWrapper, 28
NetworkDefinition, 43

openDomainConsole
 ILibvirtWrapper, 25
 LibvirtWrapper, 39
openHandlerLive
 ILibpcapWrapper, 15
 LibpcapWrapper, 28
openSnifferHandler
 PacketSniffer, 46
openVirtualMachineConsole
 VirtualizationFacade, 58
 VirtualMachineConsoleManager, 61

Packet, 44
PacketSniffer, 44
 closeAndStopListening, 45
 getNumberOfReceivedPackets, 45
 getPacketFromQueue, 45
 listenForPacket, 46
 openSnifferHandler, 46
PacketSnifferException, 47
 PacketSnifferException, 47
 what, 47

receiveDataFromConsole
 VirtualizationFacade, 58
receiveDataFromStream
 ILibvirtWrapper, 25
 LibvirtWrapper, 39
runAndObtainExecutionInfo
 ExecutionInfoObtainer, 12

sendDataToConsole
 VirtualizationFacade, 58
sendDataToStream
 ILibvirtWrapper, 25
 LibvirtWrapper, 40

VirtualMachineConsoleManager, 61
src/exceptions/PacketSnifferException.h, 71
src/exceptions/VirtualizationException.h, 71
src/interfaces/ILibpcapWrapper.h, 71
src/interfaces/ILibvirtWrapper.h, 72
src/models/ConnectionInfo.h, 73
src/models/ExecutionInfo.h, 73
src/models/ListenCallbackArgs.h, 73
src/models/NetworkDefinition.h, 73
src/models/Packet.h, 74
src/models/StreamData.h, 74
src/models/Version.h, 74
src/models/VirtualMachineInfo.h, 74
src/packetsniffer/PacketSniffer.h, 75
src/utils/ExecutionInfoObtainer.h, 75
src/utils/StringUtils.h, 75
src/utils/VersionUtils.h, 75
src/virt/managers/BaseManagerWithConnection.h, 76
src/virt/managers/ConnectionManager.h, 76
src/virt/managers/VirtualMachineConsoleManager.h, 77
src/virt/managers/VirtualMachineManager.h, 77
src/virt/managers/VirtualNetworkManager.h, 77
src/virt/VirtualizationFacade.h, 78
src/wrappers/LibpcapWrapper.h, 79
src/wrappers/LibvirtWrapper.h, 79
StreamData, 48
StringUtils, 48
 copyStringToCharArray, 48
StringUtilsTests, 49

TestingUtils, 49
tests/mocks/LibvirtWrapperMock.h, 80
tests/mocks/VirtualMachineManagerMockGetInfoAboutVirtualMachine.h, 80
tests/TestingUtils.h, 81

updateConnection
 BaseManagerWithConnection, 8
updateVmDevice
 ILibvirtWrapper, 26
 LibvirtWrapper, 40
 VirtualizationFacade, 58
 VirtualMachineManager, 65

Version, 49
VersionUtils, 50
 getVersion, 50
VersionUtilsTests, 51
VirtualizationException, 51
 VirtualizationException, 52
 what, 52
VirtualizationFacade, 52
 attachDeviceToVm, 55
 closeStream, 55
 createVirtualMachine, 55
 createVirtualNetworkFromXml, 55
 destroyNetwork, 56
 detachDeviceFromVm, 56
 getConnectionInfo, 56

getInfoAboutVirtualMachine, 56
getListOfVirtualMachinesWithInfo, 57
getNetworkDefinition, 57
initializeConnection, 57
isConnectionAlive, 57
openVirtualMachineConsole, 58
receiveDataFromConsole, 58
sendDataToConsole, 58
updateVmDevice, 58
VirtualizationFacade, 54
VirtualMachineConsoleManager, 59
BaseManagerWithConnection, 60
closeStream, 60
getDataFromStream, 60
openVirtualMachineConsole, 61
sendDataToStream, 61
VirtualMachineConsoleManagerTests, 61
VirtualMachineInfo, 62
VirtualMachineManager, 62
attachDeviceToVirtualMachine, 63
BaseManagerWithConnection, 64
createVirtualMachine, 64
detachDeviceFromVirtualMachine, 64
getInfoAboutVirtualMachine, 64
 getListOfVirtualMachinesWithInfo, 65
updateVmDevice, 65
VirtualMachineManagerMockGetInfoAboutVirtualMachine,
65
VirtualMachineManagerTests, 67
VirtualNetworkManager, 67
BaseManagerWithConnection, 68
createNetworkFromXml, 68
destroyNetwork, 69
getNetworkXmlDefinition, 69

what
PacketSnifferException, 47
VirtualizationException, 52