**vtStor Documentation**

1. ***General***

This is an open-source library written in C/C++ that allows applications to interface storage devices by sending commands.

1. ***Requirements***

vtStor provides some services:

* The architecture supports developing modules independently.
* The architecture supports extending easily.
* The architecture supports developing on multi-platform.
* Provides a managed layer for building C# applications.

1. ***User Guide***

Look at this example:

#include <memory>

#include "ErrorCodes.h"

#include "BusType.h"

#include "vtStor.h"

#include "Buffer.h"

#include "CommandHandlerAta.h"

#include "DriveAtaCommandExtensions.h"

#include "DriveEnumeratorAta.h"

#include "ProtocolAtaPassThrough.h"

#include "vtStorAta.h"

#include "Drive.h"

void main()

{

const vtStor::U32 sDefaultCommandHandlerAtaCommandType = 0;

// Initializing a drive manager which can enumerate wanted-devices pluged into the host via drive enumerators.

std::unique\_ptr<vtStor::cDriveManagerInterface> driveManager;

vtStorInit( driveManager );

// Initializing a drive enumerator which calls directly enumerating devices with a special type such as ata, scsi etc.

std::shared\_ptr<vtStor::cDriveEnumeratorAta> driveEnumeratorAta;

driveEnumeratorAta = std::make\_shared<vtStor::cDriveEnumeratorAta>();

// Drive manager registers drive enumerators.

driveManager->RegisterDriveEnumerator(driveEnumeratorAta);

// Drive manager calls enumeration. So, all drive enumerators registered will run to find out compatible devices. All device found will be listed into a vector.

driveManager->EnumerateDrives( vtStor::eScanForHardwareChanges::No );

// Drive manager gets the device vector.

vtStor::Vector\_Drives drives = driveManager->GetDrives();

// Create a data buffer of 512 bytes to use when sending commands

std::shared\_ptr<vtStor::cBufferInterface> dataBuffer = std::make\_shared<vtStor::cBuffer>(512);

// Declare protocol and command handle

// Command handlers work on Drive Layer to prepare necessary data for Protocol Layer

// Protocols work on Protocol Layer, use API functions on each OS platform to issue commands.

std::shared\_ptr<vtStor::Protocol::cProtocolInterface> protocol = nullptr;

std::shared\_ptr<vtStor::cCommandHandlerInterface> commandHandler = nullptr;

// Choose drive 1 in the device vector to use

std::shared\_ptr<vtStor::cDrive> drive = std::dynamic\_pointer\_cast<vtStor::cDrive>(drives[1]);

// Initializing protocol and command handler

protocol = std::make\_shared<vtStor::Protocol::cAtaPassThrough>();

commandHandler = std::make\_shared<vtStor::cCommandHandlerAta>(protocol);

// A drive can register one or more command handlers

drives[1]->RegisterCommandHandler(sDefaultCommandHandlerAtaCommandType, commandHandler);

// At last, call API functions to issue commands.

vtStor::Ata::IssueCommand\_IdentifyDevice(drives[1], sDefaultCommandHandlerAtaCommandType, dataBuffer);

vtStor::U8\* data = dataBuffer->ToDataBuffer();

// dump buffer data here

}