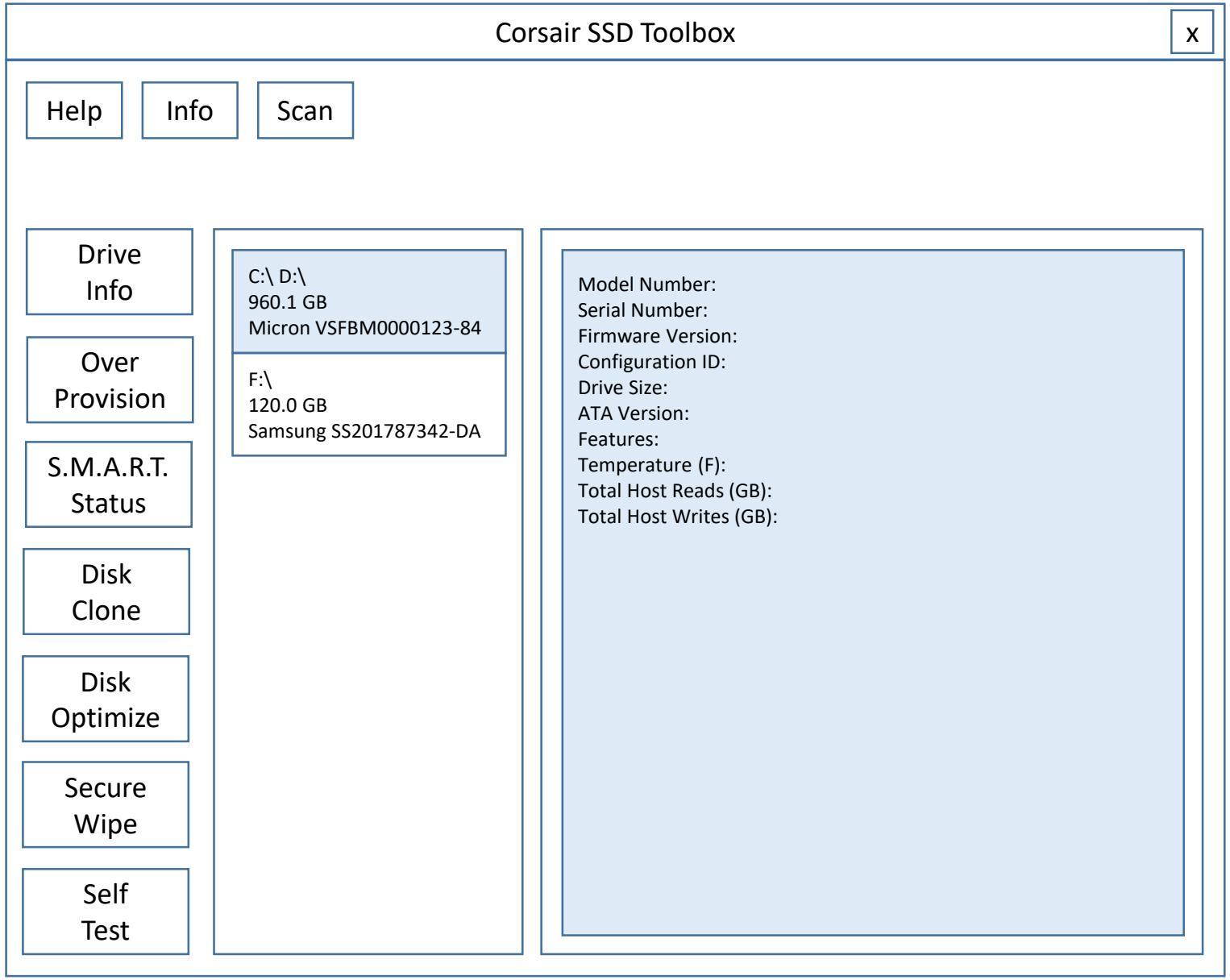
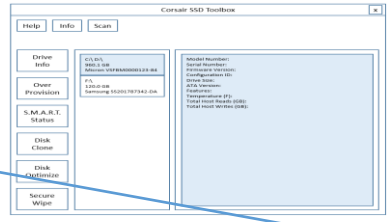
Corsair SSD Toolbox Project

Supported Features



How to access SSD in Windows



GUI (QML/C++)

We will build these software

Parameters

Direct call in C++

StorageAPI(C++, WinAPI)

ATA Command

Windows Drivers

Host Bus Adapter

SATA bus

ATA Command

SCSI bus

SCSI Command

ATA Command

ATA command embedded in SCSI command

SSD

HDD

USB Flash

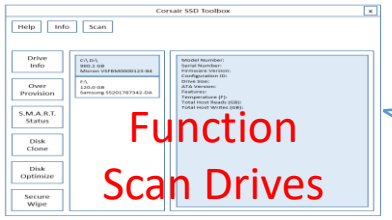
Functions of StorageAPI:

+ Handle ATA Commands
+ Provide APIs to access drives from applications
+ Utilities (access partitions, features, …)

GUI (QML/C++)

Direct call in C++

StorageAPI(C++, WinAPI)



User click "Stop"

```
void Gui::HandleStopScan() {
  if (pScanInfo) pScanInfo->stop = true;
}
```

User click "Scan"

```
void Gui::HandleScanDrive() {
  pScanInfo = new sScanInfo();
  FreezeGui();
  CreateThread(ScanDriveThreadFunc, (void*)info);
  while(1) {
    QCoreApplication::processEvent();
    if (info->done == true) break;
    ShowScanProgress(info-> progress)
  }
  ShowDriveInfo(info->dlst);
  EnableGui();
  delete pScanInfo;
}
```

BackEnd code

```
namespace StorageAPI {

enum eRetCode {
  Ok,
  ErrNoSpace,
  ErrNoMem,
  ErrNotSupport, ..
}

struct sSmartItem {
  u8 id;
  string name;
  u8 curval;
  u8 worst;
  u8 threshold;
  u32 rawval;
  u32 status;
}

struct sSmartInfo {
  vector<sSmartItem> slst;
}

struct sIdentifyInfo {
  string model
  string serial
  string version
  string confid
  string ata
  u64 cap;
  u64 feature;
  u64 thr, thw;
}

struct sDriveInfo {
  string name;
  u32 index;
  sSmartInfo smart;
  sIdentifyInfo info;
};

  eRetCode ScanDrives(vector<sDriveInfo>& dlst);
  eRetCode GetScanProgress(u8& prog)
  eRetCode UpdateFirmware(const string& drvname, u8* data, u32 size);
  eRetCode TrimDrive(const string& drvname);
  eRetCode SecureErase(const string& drvname);
  …
}
```

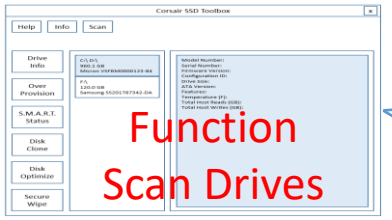
GUI code

```
struct sScanInfo {
  bool stop;
  bool done;
  u32 progress;
  vector<sDriveInfo> dlst;
};

static sScanInfo* pScanInfo;

static void ScanDriveThreadFunc(void* param) {
  sScanInfo* info = (sScanInfo*) param;
  ret = StorageApi::ScanDrives(info->dlst);
  while(!info->stop && !info->status) {
    StorageApi::GetScanProgress(info->progress);
    sleep(1ms);
  }
  info->done = true;
}
```

GUI (QML/C++)

Direct call in C++

StorageAPI(C++, WinAPI)



User click "Stop"

```
void Gui::HandleStopScan() {
  if (pScanInfo) pScanInfo->stop = true;
}
```

User click "Scan"

```
void Gui::HandleScanDrive() {
  pScanInfo = new sScanInfo();
  FreezeGui();
  CreateThread(ScanDriveThreadFunc, (void*)info);
  while(1) {
    QCoreApplication::processEvent();
    if (info->done == true) break;
    ShowScanProgress(info-> progress)
  }
  ShowDriveInfo(info->dlst);
  EnableGui();
  delete pScanInfo;
}
```

GUI code

```
void Gui::ShowDriveInfo(vector<sDriveInfo>& dlst) {
  foreach drv in dlst:
    prtstr = StorageApi::GetPartsString(drv.name)
    capstr = ToString(drv.info.cap)
    mdlstr = drv.info.model;
    drvstr = prtstr + capstr + mldstr;
    ListWidget->addItem(drvlst);
}
```

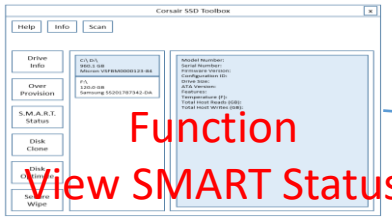
BackEnd code

```
namespace StorageAPI {

enum eRetCode {
  Ok,
  ErrNoSpace,
  ErrNoMem,
  ErrNotSupport, ..
}

struct sIdentifyInfo {
  string model
  string serial
  string version
  string confid
  string ata
  u64 cap;
  u64 feature;
  u64 thr, thw;
}

struct sSmartItem {
  u8 id;
  string name;
  u8 curval;
  u8 worst;
  u8 threshold;
  u32 rawval;
  u32 status;
}

struct sSmartInfo {
  vector<sSmartItem> slst;
}

struct sDriveInfo {
  string name;
  u32 index;
  sSmartInfo smart;
  sIdentifyInfo info;
};

  eRetCode ScanDrives(vector<sDriveInfo>& dlst);
  eRetCode GetScanProgress(u8& prog)
  eRetCode UpdateFirmware(const string& drvname, u8* data, u32 size);
  eRetCode TrimDrive(const string& drvname);
  eRetCode SecureErase(const string& drvname);
  …
}
```

GUI (QML/C++)

Direct call in C++

StorageAPI(C++, WinAPI)



User click
“SMART Status”

```
void Gui::HandleViewSmart() {
  // Display information from dlst (in Scan step)
  foreach drv in dlst:
    sSmartInfo& sm = drv.smart;
    foreach item in sm.slst:
      ShowSmartItem(row++, item);
}
```

GUI code

```
void Gui::ShowSmartItem(u32 row, sSmartIte& item) {
  TableWidget->addItem(row, col0, ToString(item.id));
  TableWidget->addItem(row, col1, item.name)
  TableWidget->addItem(row, col2, ToString(drv.curval));
  TableWidget->addItem(row, col3, ToString(drv.worst));
  TableWidget->addItem(row, col4, ToString(drv.threshold));
  TableWidget->addItem(row, col5, ToString(drv.rawval));
  TableWidget->addItem(row, col6, ToString(drv.status));
}
```

Table Widget

ID	Attribute	Current Value	Threshold	Raw Value	Worst	Status
01	Raw Read Error Rate	100	70	0	100	N/A
05	Reallocated Sector Count	100	0	0	100	N/A
09	Power On Hours (POH)	100	0	1215	100	N/A
0C	Power Cycle Count	100	0	650	100	N/A
A0	[Unknown Attribute]	100	0			N/A
A1	[Unknown Attribute]	100	0			N/A
A3	[Unknown Attribute]	100	0			N/A
A4	[Unknown Attribute]	100	0			N/A

BackEnd code

```
namespace StorageAPI {

enum eRetCode {
  Ok,
  ErrNoSpace,
  ErrNoMem,
  ErrNotSupport, ..
}

struct sIdentifyInfo {
  string model
  string serial
  string version
  string confid
  string ata
  u64 cap;
  u64 feature;
  u64 thr, thw;
}

struct sSmartItem {
  u8 id;
  string name;
  u8 curval;
  u8 worst;
  u8 threshold;
  u32 rawval;
  u32 status;
}

struct sSmartInfo {
  vector<sSmartItem> slst;
}

struct sDriveInfo {
  string name;
  u32 index;
  sSmartInfo smart;
  sIdentifyInfo info;
};

  eRetCode ScanDrives(vector<sDriveInfo>& dlst);
  eRetCode GetScanProgress(u8& prog)
  eRetCode UpdateFirmware(const string& drvname, u8* data, u32 size);
  eRetCode TrimDrive(const string& drvname);
  eRetCode SecureErase(const string& drvname);
  …
}
```

Function
Firmware Update

Use Case:
+ The user selects Corsair’s drive and clicks "Firmware Update".

GUI side:
+ Get model number and firmware version of selected drive. Discard if this is not Corsair’s drives. (*)
+ Get the updated version from Corsair’s website.
+ If there is new firmware for this drive (**), download the firmware and keep it in-memory (RAM) to prevent the end-user from modifying the firmware.
+ Verify the checksum value of the firmware. Discard it if it's not correct.
+ Call StorageApi::UpdateFirmware() with this firmware binary data.
+ Keep responsiveness to end-user’s activities.

(*) To do this, we need information about Corsair’s model string pattern, for example CRS0123XXXXX-YYY ?
(**) How to know this firmware version is newer than the other one ?
→ Confirm with PO

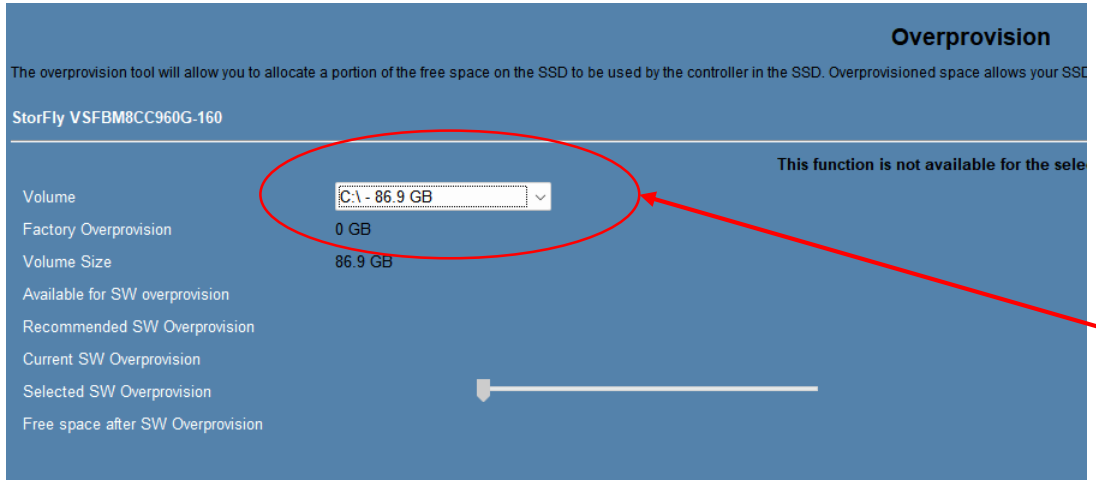
Function
Change OverProvision

User click
“Cancel”

```
void Gui::HandleStopChangeProvision() {
  if (pProvInfo) pProvInfo->stop = true;
}
```

User click
“Start”

```
void Gui::HandleStartChangeProvision() {
  // Similar to ScanDrive function
  // + Get ProvInfo(name, rate)
  // + Create Worker Thread
  // + Wait (also response to user’s activities)
  // + Finish & report status
}
```



End-user select a Partition or PhysicalDevice ?
→ Confirm with PO

It doesn’t make sense if we select a partition here, because the Over-Provision feature affects the whole drive.

Sample implementation

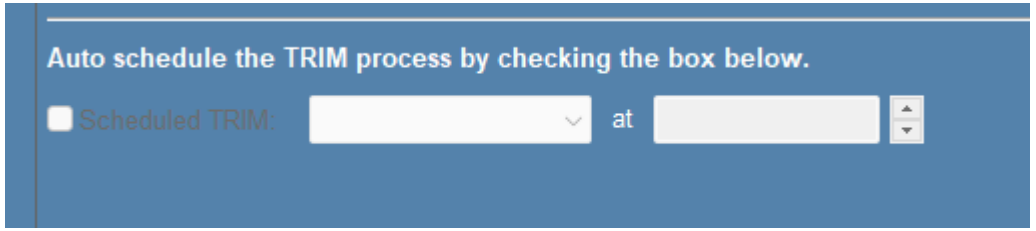
Function Optimize

User click "Cancel"

```
void Gui::HandleStopChangeProv() {
  if (pProvInfo) pProvInfo->stop = true;
}
```

User click "Start"

```
void Gui::HandleStartChangeProv() {
  // Similar to ScanDrive function
  // + Get TrimInfo(name, time)
  // + Create Worker Thread
  // + Wait (also response to user's activities)
  // + Finish & report status
}
```



User may choose to run TRIM command later on selected drive.
We may:

1. Create a background process to wait and execute TRIM later
2. Use Windows Task Scheduler to create automated task
3. Think about other solutions ?

In option 2 above, we may need to support a configuration file so that the program will be able to know which drive it will run the TRIM process on.

Sample implementation

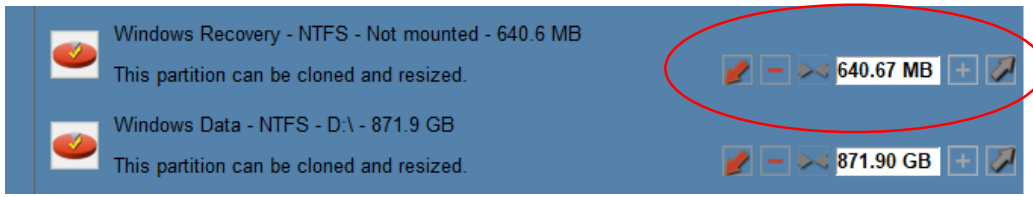
Function DiskClone

User click "Cancel"

```
void Gui::HandleStopClone() {
  if (pCloneInfo) pCloneInfo->stop = true;
}
```

User click "Start"

```
void Gui::HandleStartClone() {
  // Similar to ScanDrive function
  // + Get CloneInfo(srcdrv, parts, orders, dstdrv)
  // + Create Worker Thread
  // + Wait (also response to user's activities)
  // + Finish & report status
}
```



[Down]/[Up]: Set clone size to Min/Max
[Minus]/[Plus]: Shrink/Expand partition size in target drive
[><]: Unknown feature. Confirm with PO ?

Clarify these items with the project owner:
+ Clone full source drive or some partitions ?
+ Clone old drive to address 0 of new drive or from an offset ? Which offset ?
+ Support to merge partitions from multiple drives ? Order of these partitions in target drive ?
+ Support shrinking partition or not ?

In the case of shrinking partitions, we need to access the file system (for example, the FAT table/NTFS master file table). This complicates the code a lot. The GUI team should study about this before hand.

For simplicity, we may copy sectors one-by-one from original drive to new target drive. But it is slow. Any better solution ? Copy block of 256 sectors ? Check other tools (clonezilla) ?

Sample implementation

Function
Secure Wipe

User click
“Cancel”

```
void Gui::HandleStopSecureWipe() {
  if (pWipeInfo) pWipeInfo->stop = true;
}
```

User click
“Start”

```
void Gui::HandleStartSecureWipe() {
  // Similar to above function
}
```

Function
Self-Test

User click
“Cancel”

```
void Gui::HandleStopSelfTest() {
  if (pTestInfo) pTestInfo->stop = true;
}
```

User click
“Start”

```
void Gui::HandleStartSelfTest() {
  // Similar to above function
}
```

Source code structure

1. Put GUI and StorageAPI in the same codebase
2. Compiler: MinGW ? MSBuild ? Static ? (follow iCUE team ?)
3. In the first step, I will provide a dummy version of StorageApi with necessary data structure and “empty” implementation only. So the GUI team will be able to develop and compile GUI functions without care about underlying work. Real feature will be implemented and integrated into source code later.

```
namespace StorageAPI {

  struct sSmartItem {
    u8 id;
    string name;
    u8 curval;
    u8 worst;
    u8 threshold;
    u32 rawval;
    u32 status;
  }

  struct sSmartInfo {
    vector<sSmartItem> slst;
  }

  struct sDriveInfo {
    string name;
    u32 index;
    sSmartInfo smart;
    sIdentifyInfo info;
  };

  struct sIdentifyInfo {
    string model
    string serial
    string version
    string confid
    string ata
    u64 cap;
    u64 feature;
    u64 thr, thw;
  }

  eRetCode ScanDrives(vector<sDriveInfo>& dlst);
  eRetCode GetScanProgress(u8& prog)
  eRetCode UpdateFirmware(const string& drvname, u8* data, u32 size);
  eRetCode TrimDrive(const string& drvname);
  eRetCode SecureErase(const string& drvname);
  …
}
```

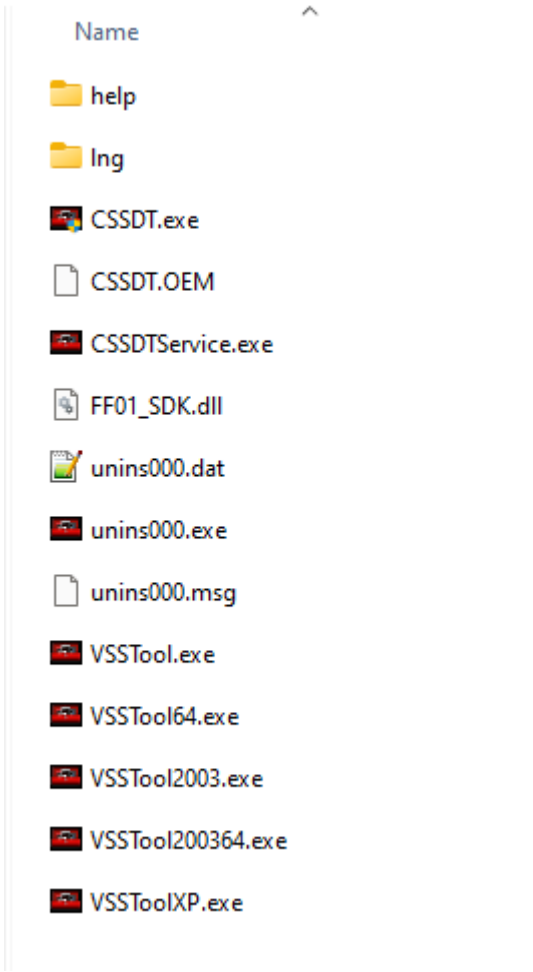
Necessary structures provided

For now, the GUI team should design and implement code to enable the end-user to start and stop a working thread, update progress information to the end-user, and keep the GUI responsive.

```
eRetCode ScanDrives(vector<sDriveInfo>& dlst) {
  for(int i = 0; i < 1000000; i++) {
    val = i * i * i;
  }
  // for now, fill dlst with fake info
  return OK;
}
```

Dummy implementation only

CSSDT installation directory



New directory structure

1. Need to deploy Qt dlls here ? (confirm with iCUE team)
2. Need to build an installer/uninstaller ? (confirm with PO)

Keywords in storage application software

+ HDD – Hard Disk Drive
+ SSD – Solid-State Drive
+ Nand Flash Memory
+ LBA: Logical Block Address.
+ Sector: 512 bytes unit.
+ Physical Drive vs Partitions.
+ Names of PhysicalDrive: /dev/sda in linux, or \\.\PhysicalDriveX in Windows (X from 0 -> 15).
+ Names of partitions: C:\, D:\
+ File system and Master file table.
+ Over-Provisioning.
+ TRIM Commands.
+ Secure-Erase/Secure-Wipe
+ S.M.A.R.T.: Self-Monitoring, Analysis, and Reporting Technology
+ SMART Attributes
+ Power Cycle

End