4.1. WINDOWS MANAGEMENT INSTRUMENTATION (WMI)

4.1.1 Introduction

There is an industry initiative that establishes management infrastructure standards and provides a way to combine information from various hardware and software management systems. This initiative is called Web-Based Enterprise Management (WBEM). WBEM is based on the Common Information Model (CIM) schema, which is an industry standard driven by the Desktop Management Task Force (DMTF).

Microsoft Windows Management Instrumentation (WMI) is an implementation of the WBEM initiative for supported Windows platforms [84]. WMI is useful in a Windows enterprise network where it reduces the maintenance and cost of managing enterprise network components. WMI provides:

- A consistent model of Windows operation, configuration, and status.
- A COM API to allow access to management information.
- The ability to operate with other Windows management services.
- A flexible and extensible architecture allowing vendors a means of writing other WMI providers to support new devices, applications, and other enhancements.
- The WMI Query Language (WQL) to create detailed queries of the information.
- An API for management application developers to write Visual Basic or Windows Scripting Host (WSH) scripts.

The WMI architecture has two parts:
1. A management infrastructure that includes the CIM Object Manager (CIMOM) and a central storage area for management data called the CIMOM object repository. CIMOM allows applications to have a uniform way to access management data [85].

2. WMI providers. WMI providers are the intermediaries between CIMOM and managed objects. Using WMI APIs, WMI providers supply CIMOM with data from managed objects, handle requests on behalf of management applications, and generate event notifications.

Windows Management Instrumentation (WMI) providers are standard COM or DCOM servers that function as mediators between managed objects and the CIM Object Manager (CIMOM). If the CIMOM receives a request from a management application for data that is not available from the CIMOM object repository, or for events, the CIMOM forwards the request to the WMI providers [86]. WMI providers supply data, and event notifications, for managed objects that are specific to their particular domain.

Windows Management Instrumentation (WMI) provides a method to configure, manage, and monitor nearly all the resources on a Windows server or workstation. It is an implementation of the Distributed Management Task Force (DMTF) Common Information Model (CIM) and Web-Based Enterprise Management (WBEM) standards.

### 4.1.2 Overview

WMI was first introduced to Windows in 1998 with Windows NT 4.0 Service Pack 4 and is available in all subsequent Windows versions. It provides a method for managing nearly all Windows components locally or remotely with a variety of tools including scripting languages and a command line tool called WMIC (Windows Management Instrumentation Command-line) [87]. It provides an interface to file systems, event logs, devices, services, hardware controllers, processing, memory, user accounts, and many other aspects of the
Windows operating system and installed applications. Components and applications that can be managed by WMI are made available through DLLs and are known as “providers”.

4.1.3 Features

For someone willing to develop one or many WMI providers, WMI offers many features out of the box. Here are the most important advantages:

1. **Automation interfaces**: Because WMI comes with a set of automation interfaces ready to use, all management features supported by a WMI provider and its set of classes get the scripting support for free out-of-the-box. Beyond the WMI class design and the provider development, the Microsoft development and test teams are not required to create, validate and test a scripting model as it is already available from WMI.

2. **.NET Management interfaces**: Because the System. Management namespace relies on the existing COM/DCOM plumbing, the created WMI provider and its set of WMI classes becomes automatically available to all .NET applications independently of the language used (e.g. C#, VB.NET). Beyond the WMI class design and the provider development, like for scripting, the Microsoft development and test teams are not required to create, validate and test new assemblies to support a new namespace in the .NET Framework as this support is already available from WMI for free.

3. **C/C++ COM/DCOM programming interfaces**: Like most components in Windows, COM/DCOM programmers can leverage the features of the provider they develop at the COM/DCOM interfaces level. Like in previous environments (scripting and .NET Framework), a COM/DCOM consumer just needs to interact with the standard set of WMI COM interfaces to leverage the WMI provider capabilities and its set of supported WMI classes. To make all management information available from the native APIs, the
WMI provider developer just needs to interact with a set of pre-defined WMI COM interfaces. This will make the management information available at the WMI COM level automatically. Moreover, the scripting COM interface object model is very similar to the COM/DCOM interface object model, which makes it easy for developers to be familiar with the scripting experience.

4. **Remoting capabilities over DCOM and SOAP**: More than simply offering local COM capabilities, as management is all about remoting, WMI offers the DCOM transport. In addition, SOAP transport will be available in Windows Server 2003 R2 through the WS-Management initiative led by Microsoft, Intel, Sun Microsystems and Dell. This initiative allows to run any scripts remotely or to consume WMI data through a specific set of interfaces handling SOAP requests/responses [88]. The advantage for the WMI provider developer is that when he exposes all his features through WMI, *Windows Remote Management/WS-Management* can in turn consume that information as well (embedded objects in WMI instances are not supported in Windows Server 2003 R2. It is however a target for Vista). All the layering to WS-Management and the mapping of the CIM data model to SOAP comes for free out of the WMI/WS-Management solution. In the event DCOM must be used, implementing DCOM requires the presence of a proxy DLL deployed on each client machine. As WMI is available in the Windows operating system since Windows 2000, these issues are eliminated.

5. **Support for Queries**: WMI offers support for WQL queries out of the box. This means that if a provider is not designed to support queries, WMI supports it by using an enumeration technique out of the provider.

6. **Eventing capabilities**: WMI offers the capability to notify a subscriber for any event it is interested in. WMI uses the WMI Query Language (WQL) to submit WQL event queries
and defines the type of events to be returned [89]. The evening mechanism, with all related callbacks, is part of the WMI COM/DCOM and automation interfaces. Anyone writing a WMI provider can have the benefit of this functionality at no cost for his customers. It will be up to the consumer to decide how it wants to consume the management information exposed by the WMI provider and its related set of WMI classes.

7. **Code template generator:** To speed up the process of writing a WMI provider including all COM/DCOM interfaces and related definitions, the WMI team developed the *WMI ATL Wizard* to generate the code template implementing a provider. The code generated is based on the WMI class model initially designed by the developer. The WMI provider developer will be able to interface the pre-defined COM/DCOM interfaces for the WMI provider with its set of native APIs retrieving the management information to expose. The exercise consists in filling the “gaps” in the provider code to create the desired interfacing logic.

8. **Predictability:** Predictability is an important concern for IT professionals because it defines the capability of someone having an experience with a set of interfaces managing a Windows component to apply this knowledge right away, intuitively, to any other manageable Windows component without having relearn everything from ground up. Predictability for a customer is a real gain as it increases the Return of Investment (ROI). A person facing such a situation simply expects things to work the same way based on his previous experience. The constant increase of COM programming/scriptable interfaces has a huge impact on the predictability, as this makes it difficult for customers to automate, manage Windows and leverage their existing knowledge. WMI with CIM
address this problem by always exposing the same programming object model (COM/DCOM, Automation, .NET) whatever the manageable entity is.

9. **Protect existing customer investments:** Protecting customers and partners investment motivates customers to invest in technologies. As Microsoft did invest a lot these past years in writing WMI providers, customers and partners invested in tools leveraging the WMI capabilities of Windows. Therefore, they naturally continue to exploit these capabilities instead of having to use a new set of specific interfaces for each Windows manageable component. A specific set of interfaces means having a specific set of agents or in-house developed software based on a new model or set of interfaces especially dedicated to a component or technology. By leveraging the capabilities of WMI today, customers and partners can leverage the work investment made in the past while minimizing their costs in developments, learning curves and new discoveries. This will also have a great impact on the stability and reliability of their infrastructure as they continue to leverage an existing implementation with an improved technology.

10. **Provide a logical and unified administration model:** As briefly described before in the introduction, this model is based on an industry standard called CIM defined by the DMTF (http://www.dmtf.org). The CIM class-based schema is defined by a consortium of constructors and software developers that meets the requirements of the industry. This implies that not only Microsoft leverages the WMI capabilities, but also any other third party constructors or developers write their own code to fit into the model. For instance, Intel is doing this for some their network driver adapters and software. HP is leveraging existing WMI providers and implementing their own WMI providers in their HP Open View Enterprise Management software. IBM consumes WMI from the Tivoli management suite, MOM and SMS are also consuming and providing WMI information.
Lastly, Windows XP SP2 leverages WMI to get information status from anti-virus software and firewalls.

4.1.4 WMI Tools

Some WMI tools can also be useful during the design and development phases. These tools are:

- **The MOF compiler (MOFComp.exe):** The Managed Object Format (MOF) compiler parses a file containing Managed Object Format statements and adds the classes and class instances defined in the file to the CIM repository. The MOF format is a specific syntax to define CIM class representation in an ASCII file (e.g. MIB are to SNMP what MOF files are to CIM). MOFComp.exe is included in every WMI installation [90]. Every definition existing in the CIM repository is initially defined in an MOF file. MOF files are located in %System Root%\System32\WBEM. During the WMI setup, they are loaded in the CIM repository.

- **The WMI Administrative Tools:** The WMI Administrative Tools are made of four tools: WMI CIM Studio, WMI Object Browser, WMI Event Registration and WMI Event Viewer. WMI Administrative Tools can be downloaded here. The most important tool for a WMI provider developer is WMI CIM Studio as it helps in the initial WMI class creation in the CIM repository [91]. It uses a web interface to display information and relies on a collection of ActiveX components installed on the system when it runs for the first time. WMI CIM Studio provides the ability to:
  - Connect to a chosen system and browse the CIM repository in any namespace available.
  - Search for classes by their name, by their descriptions or by property names.
• Review the properties, methods and associations related to a given class.

• See the instances available for a given class of the examined system.

• Perform Queries in the WQL language.

• Generate an MOF file based on selected classes.

• Compile an MOF file to load it in the CIM repository.

• **WinMgmt.exe:** WinMgmt.exe is not a tool; it is the executable that implements the WMI Core service. Under the Windows NT family of operating systems, WMI runs as a service. On computers running Windows 98, Windows 95 or Windows Me, WMI runs as an application. Under the Windows NT family of operating systems, it is also possible to run this executable as an application, in which case, the executable runs in the current user context. For this, the WMI service must be stopped first. The executable supports some switches that can be useful when starting WMI as a service or as an application [92]. WMI provider developers who may want to debug their providers essentially need to run the WMI service as an application.

• **WBEMTest.exe:** WBEMTest.exe is a WMI tester tool, which is delivered with WMI. This tool allows an administrator or a developer to perform most of the tasks from a graphical interface that WMI provides at the API level. Although available under all Windows NT-based operating systems, this tool is not officially supported by Microsoft. WBEMTest provides the ability to:

  • Enumerate, open, create and delete classes.
  
  • Enumerate, open, create and delete instances of classes.
  
  • Select a namespace.
  
  • Perform data and event queries.
  
  • Execute methods associated to classes or instances.
- Execute every WMI operation asynchronously, synchronously or semi-asynchronously.

- **The WMI command line tool (WMIC):** WMIC is a command-line tool designed to ease WMI information retrieval about a system by using some simple keywords (aliases). WMIC.exe is only available under Windows XP Professional, Windows Server 2003, Windows Vista and Windows Server 2008. By typing “WMIC /?” from the command-line, a complete list of the switches and reserved keywords is available.
  - There is a Linux port of WMI command line tool, written in Python, based on Samba4 called 'wmi-client'

- **WBEMDump.exe:** WBEMDump is a tool delivered with the Platform SDK. This command line tool comes with its own Visual C++ project. The tool can show the CIM repository classes, instances, or both. It is possible to retrieve the same information as that retrieved with WMIC. WBEMDump.exe requires more specific knowledge about WMI, as it doesn’t abstract WMI as WMIC. However, it runs under Windows NT 4.0 and Windows 2000. It is also possible to execute methods exposed by classes or instances [93]. Even if it is not a standard WMI tool delivered with the system installation, this tool can be quite useful for exploring the CIM repository and WMI features.

**4.1.4.1 Win32_DiskDrive Class**

The Win32_DiskDrive WMI class represents a physical disk drive as seen by a computer running the Windows operating system. Any interface to a Windows physical disk drive is a descendent (or member) of this class. The features of the disk drive seen through this object correspond to the logical and management characteristics of the drive [94]. In some cases, this may not reflect the actual physical characteristics of the device. Any object based on another logical device would not be a member of this class. The following syntax is simplified from
Managed Object Format (MOF) code and includes all of the inherited properties [95]. Properties are listed in alphabetic order, not MOF order.

**Syntax:**

```csharp
class Win32_DiskDrive : CIM_DiskDrive {
    uint16   Availability;
    uint32   BytesPerSector;
    uint16   Capabilities[];
    string   CapabilityDescriptions[];
    string   Caption;
    string   CompressionMethod;
    uint32   ConfigManagerErrorCode;
    boolean  ConfigManagerUserConfig;
    string   CreationClassName;
    uint64   DefaultBlockSize;
    string   Description;
    string   DeviceID;
    boolean  ErrorCleared;
    string   ErrorDescription;
    string   ErrorMethodology;
    string   FirmwareRevision;
    uint32   Index;
    datetime InstallDate;
    string   InterfaceType;
    uint32   LastErrorCode;
    string   Manufacturer;
    uint64   MaxBlockSize;
    uint64   MaxMediaSize;
    boolean  MediaLoaded;
    string   MediaType;
    uint64   MinBlockSize;
```
string Model;
string Name;
boolean NeedsCleaning;
uint32 NumberOfMediaSupported;
uint32 Partitions;
string PNPDeviceID;
uint16 PowerManagementCapabilities[];
boolean PowerManagementSupported;
uint32 SCSIBus;
uint16 SCSILogicalUnit;
uint16 SCSIPort;
uint16 SCSITargetId;
uint32 SectorsPerTrack;
string SerialNumber;
uint32 Signature;
uint64 Size;
string Status;
uint16 StatusInfo;
string SystemCreationClassName;
string SystemName;
uint64 TotalCylinders;
uint32 TotalHeads;
uint64 TotalSectors;
uint64 TotalTracks;
uint32 TracksPerCylinder;

};

Methods

The Win32_DiskDrive class has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>It requests a reset of the logical device. The return value should be 0 if the request was successfully executed, 1 if the request is not supported and some</td>
</tr>
</tbody>
</table>
It defines the desired power state for a logical device and when a device should be put into that state. The desired power state is specified by setting the PowerState parameter to one of the following integer values: 1="Full Power", 2="Power Save - Low Power Mode", 3="Power Save - Standby", 4="Power Save - Other", 5="Power Cycle" or 6="Power Off". The Time parameter (for all state changes, except 5, "Power Cycle") indicates when the power state should be set, either as a regular date-time value or as an interval value (where the interval begins when the method invocation is received). When the PowerState parameter is equal to 5, "Power Cycle", the Time parameter indicates when the device should power on again. Power off is immediate. SetPowerState should return 0 if successful, 1 if the specified PowerState and Time request is not supported, and some other value if any other error occurred. In a subclass, the set of possible return codes could be specified, using a ValueMap qualifier on the method. The strings to which the ValueMap contents are 'translated' may also be specified in the subclass as a Values array.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetPowerState</td>
<td>It defines the desired power state for a logical device and when a device should be put into that state. The desired power state is specified by setting the PowerState parameter to one of the following integer values: 1=&quot;Full Power&quot;, 2=&quot;Power Save - Low Power Mode&quot;, 3=&quot;Power Save - Standby&quot;, 4=&quot;Power Save - Other&quot;, 5=&quot;Power Cycle&quot; or 6=&quot;Power Off&quot;. The Time parameter (for all state changes, except 5, &quot;Power Cycle&quot;) indicates when the power state should be set, either as a regular date-time value or as an interval value (where the interval begins when the method invocation is received). When the PowerState parameter is equal to 5, &quot;Power Cycle&quot;, the Time parameter indicates when the device should power on again. Power off is immediate. SetPowerState should return 0 if successful, 1 if the specified PowerState and Time request is not supported, and some other value if any other error occurred. In a subclass, the set of possible return codes could be specified, using a ValueMap qualifier on the method. The strings to which the ValueMap contents are 'translated' may also be specified in the subclass as a Values array.</td>
</tr>
</tbody>
</table>

**Table 4.1: Win32_DiskDrive Methods**

**Properties**

The Win32_DiskDrive class has the following properties.

- **MediaLoaded** - It determines whether the media for a disk drive is loaded. If True, the media for a disk drive is loaded, which means that the device has a readable file system and is accessible. For fixed disk drives, this property will always be TRUE.

- **MediaType** - It is the type of media used or accessed by this device. Example: Removable media

**4.1.4.2 Win32_LogicalDisk Class**

The Win32_LogicalDisk WMI class represents a data source that resolves to an actual local storage device on a computer system running Windows.
The following syntax is simplified from Managed Object Format (MOF) code and includes all of the inherited properties [96]. Properties are listed in alphabetic order, not MOF order.

**Syntax**

```csharp
class Win32_LogicalDisk : CIM_LogicalDisk
{
    uint16   Access;
    uint16   Availability;
    uint64   BlockSize;
    string   Caption;
    boolean  Compressed;
    uint32   ConfigManagerErrorCode;
    boolean  ConfigManagerUserConfig;
    string   CreationClassName;
    string   Description;
    string   DeviceID;
    uint32   DriveType;
    boolean  ErrorCleared;
    string   ErrorDescription;
    string   ErrorMethodology;
    string   FileSystem;
    uint64   FreeSpace;
    datetime InstallDate;
    uint32   LastErrorCode;
    uint32   MaximumComponentLength;
    uint32   MediaType;
    string   Name;
    uint64   NumberOfBlocks;
    string   PNPDeviceID;
    uint16   PowerManagementCapabilities[];
    boolean  PowerManagementSupported;
}
```
Methods

The Win32_LogicalDisk class has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chkdsk</td>
<td>It invokes the chkdsk operation on the disk. The method is applicable to only those instances of logical disk that represent a physical disk in the machine. It is not applicable to mapped logical drives. The return value of the method will indicate one of the following - Success - Chkdsk completed, Success - Locked and chkdsk scheduled on reboot, Failure - Unknown file system, Failure - Unknown error, Failure - Unsupported File System.</td>
</tr>
<tr>
<td>ExcludeFromAutochk</td>
<td>It is used exclude disks from the chkdsk operation to be run at the next reboot. If not excluded, chkdsk is performed on the disk if the dirty bit has been set for the disk. Note that the</td>
</tr>
</tbody>
</table>
calls to exclude disks are not cumulative. That is, if a call is made to excluded some disks, then the new list is not added to the list of ones that were already marked for exclusion, instead the new list of disks overwrites the previous one. The method is applicable to only those instances of logical disk that represent a physical disk in the machine and is not applicable to mapped logical drives. For example, a valid specification of drives would be "C:", "d:","G:" Note that the colon is required with the drive letter.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reset</strong></td>
<td>It requests a reset of the logical device. The return value should be 0 if the request was successfully executed, 1 if the request is not supported and some other value if an error occurred.</td>
</tr>
<tr>
<td><strong>ScheduleAutoChk</strong></td>
<td>It is used schedule chkdsk to be run at the next reboot if the dirty bit has been set. The method is applicable to only those instances of logical disk that represent a physical disk in the machine. It is not applicable to mapped logical drives.</td>
</tr>
<tr>
<td><strong>SetPowerState</strong></td>
<td>It defines the desired power state for a logical device and when a device should be put into that state. The desired power state is specified by setting the PowerState parameter to one of the following integer values: 1=&quot;Full Power&quot;, 2=&quot;Power Save - Low Power Mode&quot;, 3=&quot;Power Save - Standby&quot;, 4=&quot;Power Save - Other&quot;, 5=&quot;Power Cycle&quot; or 6=&quot;Power Off&quot;. The Time parameter (for all state changes, except 5, &quot;Power Cycle&quot;) indicates when the power state should be set, either as a regular date-time value or as an interval value (where the interval begins when the method invocation is received). When the PowerState parameter is equal to 5, &quot;Power Cycle&quot;, the Time parameter indicates when the device should power on again. Power off is immediate. SetPowerState should return 0 if successful, 1 if the specified PowerState and Time</td>
</tr>
</tbody>
</table>
request is not supported, and some other value if any other error occurred. In a subclass, the set of possible return codes could be specified, using a ValueMap qualifier on the method. The strings to which the ValueMap contents are 'translated' may also be specified in the subclass as a Values array qualifier.

Table 4.2: Win32_LogicalDisk Methods

Properties

The Win32_LogicalDisk class has the following properties.

- **Drive Type** - It contains a numeric value corresponding to the type of disk drive this logical disk represents. Please refer to the Platform SDK documentation for additional values. Example: A CD-ROM drive would return 5
  
  A USB drive would return 2.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unknown</td>
</tr>
<tr>
<td>1</td>
<td>No Root Directory</td>
</tr>
<tr>
<td>2</td>
<td>Removable Disk</td>
</tr>
<tr>
<td>3</td>
<td>Local Disk</td>
</tr>
<tr>
<td>4</td>
<td>Network Drive</td>
</tr>
<tr>
<td>5</td>
<td>Compact Disk</td>
</tr>
<tr>
<td>6</td>
<td>RAM Disk</td>
</tr>
</tbody>
</table>

Table 4.3: Win32_LogicalDisk Drive Type

- **FileSystem** - It indicates the file system on the logical disk. Example: "NTFS"

- **MediaType** - It indicates the type of media currently present in the logical drive. This value will be one of the values of the MEDIA_TYPE enumeration defined in winioctl.h.
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unknown</td>
</tr>
<tr>
<td>1</td>
<td>F5_1Pt2_512 5 1/4-Inch Floppy Disk - 1.2 MB - 512 bytes/sector</td>
</tr>
<tr>
<td>2</td>
<td>F3_1Pt44_512 3 1/2-Inch Floppy Disk - 1.44 MB - 512 bytes/sector</td>
</tr>
<tr>
<td>3</td>
<td>F3_2Pt88_512 3 1/2-Inch Floppy Disk - 2.88 MB - 512 bytes/sector</td>
</tr>
<tr>
<td>4</td>
<td>F3_20Pt8_512 3 1/2-Inch Floppy Disk - 20.8 MB - 512 bytes/sector</td>
</tr>
<tr>
<td>5</td>
<td>F3_720_512 3 1/2-Inch Floppy Disk - 720 KB - 512 bytes/sector</td>
</tr>
<tr>
<td>6</td>
<td>F5_360_512 5 1/4-Inch Floppy Disk - 360 KB - 512 bytes/sector</td>
</tr>
<tr>
<td>7</td>
<td>F5_320_512 5 1/4-Inch Floppy Disk - 320 KB - 512 bytes/sector</td>
</tr>
<tr>
<td>8</td>
<td>F5_320_1024 5 1/4-Inch Floppy Disk - 320 KB - 1024 bytes/sector</td>
</tr>
<tr>
<td>9</td>
<td>F5_180_512 5 1/4-Inch Floppy Disk - 180 KB - 512 bytes/sector</td>
</tr>
<tr>
<td>10</td>
<td>F5_160_512 5 1/4-Inch Floppy Disk - 160 KB - 512 bytes/sector</td>
</tr>
<tr>
<td>11</td>
<td>Removable media other than floppy</td>
</tr>
<tr>
<td>12</td>
<td>Fixed hard disk media</td>
</tr>
<tr>
<td>13</td>
<td>F3_120M_512 3 1/2-Inch Floppy Disk - 120 MB - 512 bytes/sector</td>
</tr>
<tr>
<td>14</td>
<td>F3_640_512 3 1/2-Inch Floppy Disk - 640 KB - 512 bytes/sector</td>
</tr>
<tr>
<td>15</td>
<td>F5_640_512 5 1/4-Inch Floppy Disk - 640 KB - 512 bytes/sector</td>
</tr>
<tr>
<td>16</td>
<td>F5_720_512 5 1/4-Inch Floppy Disk - 720 KB - 512 bytes/sector</td>
</tr>
<tr>
<td>17</td>
<td>F3_1Pt2_512 3 1/2-Inch Floppy Disk - 1.2 MB - 512 bytes/sector</td>
</tr>
<tr>
<td>18</td>
<td>F3_1Pt23_1024 3 1/2-Inch Floppy Disk - 1.23 MB - 1024 bytes/sector</td>
</tr>
<tr>
<td>19</td>
<td>F5_1Pt23_1024 5 1/4-Inch Floppy Disk - 1.23 MB - 1024 bytes/sector</td>
</tr>
<tr>
<td>20</td>
<td>F3_128Mb_512 3 1/2-Inch Floppy Disk - 128 MB - 512 bytes/sector</td>
</tr>
<tr>
<td>21</td>
<td>F3_230Mb_512 3 1/2-Inch Floppy Disk - 230 MB - 512 bytes/sector</td>
</tr>
<tr>
<td>22</td>
<td>F8_256_128 8-Inch Floppy Disk - 256 KB - 128 bytes/sector</td>
</tr>
</tbody>
</table>

**Table 4.4**: Win32_LogicalDisk Media Type
• **VolumeName**- It indicates the volume name of the logical disk (Constraints: Maximum 32 characters).

• **VolumeSerialNumber**- It indicates the volume serial number of the logical disk (Constraints: Maximum 11 characters). Example: "A8C3-D032"

### 4.1.4.3 Win32_OperatingSystem Class

The Win32_OperatingSystem WMI class represents a Windows-based operating system installed on a computer. Any operating system that can be installed on a computer that can run a Windows-based operating system is a descendant or member of this class. Win32_OperatingSystem is a singleton class [97]. The following syntax is simplified from Managed Object Format (MOF) code and includes all of the inherited properties. Properties and methods are in alphabetic order, not MOF order.

**Syntax:**

```plaintext
class Win32_OperatingSystem : CIM_OperatingSystem
{
    string BootDevice;
    string BuildNumber;
    string BuildType;
    string Caption;
    string CodeSet;
    string CountryCode;
    string CreationClassName;
    string CSCreationClassName;
    string CSDVersion;
    string CSName;
    sint16 CurrentTimeZone;
    boolean DataExecutionPrevention_Available;
    boolean DataExecutionPrevention_32BitApplications;
    boolean DataExecutionPrevention_Drivers;
}
```
uint8 DataExecutionPrevention_SupportPolicy;
boolean Debug;
string Description;
boolean Distributed;
uint32 EncryptionLevel;
uint8 ForegroundApplicationBoost;
uint64 FreePhysicalMemory;
uint64 FreeSpaceInPagingFiles;
uint64 FreeVirtualMemory;
datetime InstallDate;
uint32 LargeSystemCache;
datetime LastBootUpTime;
datetime LocalDateTime;
string Locale;
string Manufacturer;
uint32 MaxNumberOfProcesses;
uint64 MaxProcessMemorySize;
string MUILanguages[];
string Name;
uint32 NumberOfLicensedUsers;
uint32 NumberOfProcesses;
uint32 NumberOfUsers;
uint32 OperatingSystemSKU;
string Organization;
string OSArchitecture;
uint32 OSLanguage;
uint32 OSProductSuite;
uint16 OSType;
string OtherTypeDescription;
Boolean PAEEnabled;
string PlusProductID;
string PlusVersionNumber;
boolean Primary;
uint32 ProductType;
string RegisteredUser;
string SerialNumber;
uint16 ServicePackMajorVersion;
uint16 ServicePackMinorVersion;
uint64 SizeStoredInPagingFiles;
string Status;
uint32 SuiteMask;
string SystemDevice;
string SystemDirectory;
string SystemDrive;
uint64 TotalSwapSpaceSize;
uint64 TotalVirtualMemorySize;
uint64 TotalVisibleMemorySize;
string Version;
string WindowsDirectory;

};

Methods

The Win32_OperatingSystem class has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reboot</td>
<td>It shuts down the computer system, then restarts it. On computers running Windows NT/2000, the calling process must have the SE_SHUTDOWN_NAME privilege. The method returns an integer value that can be interpreted as follows: 0 - Successful completion. Other - For integer values other than those listed above.</td>
</tr>
</tbody>
</table>
It sets the current system time on the computer. On computer systems running Windows NT/2000, the calling process must have the SE_SYSTEMTIME_NAME privilege. The method returns an integer value that can be interpreted as follows: 0 - Successful completion. Other - For integer values other than those listed above; refer to Win32 error code documentation.

It unloads programs and DLLs to the point where it is safe to turn off the computer. All file buffers are flushed to disk, and all running processes are stopped. On computer systems running Windows NT/2000, the calling process must have the SE_SHUTDOWN_NAME privilege. The method returns an integer value that can be interpreted as follows: 0 - Successful completion. Other - For integer values other than those listed above; refer to Win32 error code documentation.

It provides the full set of shutdown options supported by Win32 operating systems. The method returns an integer value that can be interpreted as follows: 0 - Successful completion. Other - For integer values other than those listed above.

Table 4.5: Win32_OperatingSystem Class Methods

Properties

The Win32_OperatingSystem class has the following properties.

- **BootDevice** - It indicates the name of the disk drive from which the Win32 operating system boots. Example: `\Device\Harddisk0, \Device\Live-USB`.

- **BuildNumber** - It indicates the build number of the operating system. It can be used for more precise versioning information than product release version numbers. Example: 1381
- **BuildType**: It indicates the type of build used for the operating system. Examples include retail build and checked build.

- **OSArchitecture**: Architecture of the operating system, as opposed to the processor. Example: 32-bit

- **OSType**: A integer indicating the type of operating system.

The following list identifies the possible values.

<table>
<thead>
<tr>
<th>Values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0x0)</td>
<td>Unknown</td>
</tr>
<tr>
<td>1 (0x1)</td>
<td>Other</td>
</tr>
<tr>
<td>2 (0x2)</td>
<td>MACROS</td>
</tr>
<tr>
<td>3 (0x3)</td>
<td>ATTUNIX</td>
</tr>
<tr>
<td>4 (0x4)</td>
<td>DGUX</td>
</tr>
<tr>
<td>5 (0x5)</td>
<td>DECNT</td>
</tr>
<tr>
<td>6 (0x6)</td>
<td>Digital UNIX</td>
</tr>
<tr>
<td>7 (0x7)</td>
<td>OpenVMS</td>
</tr>
<tr>
<td>8 (0x8)</td>
<td>HPUX</td>
</tr>
<tr>
<td>9 (0x9)</td>
<td>AIX</td>
</tr>
<tr>
<td>10 (0xA)</td>
<td>MVS</td>
</tr>
<tr>
<td>11 (0xB)</td>
<td>OS400</td>
</tr>
<tr>
<td>12 (0xC)</td>
<td>OS/2</td>
</tr>
<tr>
<td>13 (0xD)</td>
<td>JavaVM</td>
</tr>
<tr>
<td>14 (0xE)</td>
<td>MSDOS</td>
</tr>
<tr>
<td>15 (0xF)</td>
<td>WIN3x</td>
</tr>
<tr>
<td>16 (0x10)</td>
<td>WIN95</td>
</tr>
<tr>
<td>17 (0x11)</td>
<td>WIN98</td>
</tr>
<tr>
<td>18 (0x12)</td>
<td>WINNT</td>
</tr>
<tr>
<td>19 (0x13)</td>
<td>WINCE</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>20 (0x14)</td>
<td>NCR3000</td>
</tr>
<tr>
<td>21 (0x15)</td>
<td>NetWare</td>
</tr>
<tr>
<td>22 (0x16)</td>
<td>OSF</td>
</tr>
<tr>
<td>23 (0x17)</td>
<td>DC/OS</td>
</tr>
<tr>
<td>24 (0x18)</td>
<td>Reliant UNIX</td>
</tr>
<tr>
<td>25 (0x19)</td>
<td>SCO UnixWare</td>
</tr>
<tr>
<td>26 (0x1A)</td>
<td>SCO OpenServer</td>
</tr>
<tr>
<td>27 (0x1B)</td>
<td>Sequent</td>
</tr>
<tr>
<td>28 (0x1C)</td>
<td>IRIX</td>
</tr>
<tr>
<td>29 (0x1D)</td>
<td>Solaris</td>
</tr>
<tr>
<td>30 (0x1E)</td>
<td>SunOS</td>
</tr>
<tr>
<td>31 (0x1F)</td>
<td>U6000</td>
</tr>
<tr>
<td>32 (0x20)</td>
<td>ASERIES</td>
</tr>
<tr>
<td>33 (0x21)</td>
<td>TandemNSK</td>
</tr>
<tr>
<td>34 (0x22)</td>
<td>TandemNT</td>
</tr>
<tr>
<td>35 (0x23)</td>
<td>BS2000</td>
</tr>
<tr>
<td>36 (0x24)</td>
<td>LINUX</td>
</tr>
<tr>
<td>37 (0x25)</td>
<td>Lynx</td>
</tr>
<tr>
<td>38 (0x26)</td>
<td>XENIX</td>
</tr>
<tr>
<td>39 (0x27)</td>
<td>VM/ESA</td>
</tr>
<tr>
<td>40 (0x28)</td>
<td>Interactive UNIX</td>
</tr>
<tr>
<td>41 (0x29)</td>
<td>BSDUNIX</td>
</tr>
<tr>
<td>42 (0x2A)</td>
<td>FreeBSD</td>
</tr>
<tr>
<td>43 (0x2B)</td>
<td>NetBSD</td>
</tr>
<tr>
<td>44 (0x2C)</td>
<td>GNU Hurd</td>
</tr>
<tr>
<td>45 (0x2D)</td>
<td>OS9</td>
</tr>
<tr>
<td>46 (0x2E)</td>
<td>MACH Kernel</td>
</tr>
<tr>
<td>47 (0x2F)</td>
<td>Inferno</td>
</tr>
<tr>
<td>48 (0x30)</td>
<td>QNX</td>
</tr>
<tr>
<td>No.</td>
<td>Value</td>
</tr>
<tr>
<td>-----</td>
<td>--------</td>
</tr>
<tr>
<td>49</td>
<td>0x31</td>
</tr>
<tr>
<td>50</td>
<td>0x32</td>
</tr>
<tr>
<td>51</td>
<td>0x33</td>
</tr>
<tr>
<td>52</td>
<td>0x34</td>
</tr>
<tr>
<td>53</td>
<td>0x35</td>
</tr>
<tr>
<td>54</td>
<td>0x36</td>
</tr>
<tr>
<td>55</td>
<td>0x37</td>
</tr>
<tr>
<td>56</td>
<td>0x38</td>
</tr>
<tr>
<td>57</td>
<td>0x39</td>
</tr>
</tbody>
</table>

**Table 4.6: Win32_OperatingSystem Types**

- **SystemDevice** - It indicates physical disk partition on which the operating system is installed.

- **SystemDrive** - It contains letter of the disk drive on which the operating system resides. Example: "C:", “F:” etc

- **TotalSwapSpaceSize** - Total swap space in kilobytes. This value may be **NULL** (unspecified) if the swap space is not distinguished from page files. However, some operating systems distinguish these concepts [98]. For example, in UNIX, whole processes can be swapped out when the free page list falls and remains below a specified amount.

- **WindowsDirectory** - It indicates the Windows directory of the operating system. Example: C:\WINDOWS

**4.1.4.4 Win32_NetworkAdapter Class**

The Win32_NetworkAdapter WMI class represents a network adapter of a computer running a Windows operating system. Win32_NetworkAdapter only supplies IPv4 data. For more information, see IPv6 and IPv4 Support in WMI [99]. The following syntax is simplified
from Managed Object Format (MOF) code and includes all of the inherited properties. Properties
are listed in alphabetic order, not MOF order.

**Syntax**

```csharp
class Win32_NetworkAdapter : CIM_NetworkAdapter
{
    string   AdapterType;
    uint16   AdapterTypeID;
    boolean  AutoSense;
    uint16   Availability;
    string   Caption;
    uint32   ConfigManagerErrorCode;
    boolean  ConfigManagerUserConfig;
    string   CreationClassName;
    string   Description;
    string   DeviceID;
    boolean  ErrorCleared;
    string   ErrorDescription;
    string   GUID;
    uint32   Index;
    datetime InstallDate;
    boolean  Installed;
    uint32   InterfaceIndex;
    uint32   LastErrorCode;
    string   MACAddress;
    string   Manufacturer;
    uint32   MaxNumberControlled;
    uint64   MaxSpeed;
    string   Name;
        string   NetConnectionID;
        uint16   NetConnectionStatus;
    boolean  NetEnabled;
```
string NetworkAddresses[];
string PermanentAddress;
    boolean PhysicalAdapter;
string PNPDeviceID;
uint16 PowerManagementCapabilities[];
boolean PowerManagementSupported;
string ProductName;
string ServiceName;
uint64 Speed;
string Status;
uint16 StatusInfo;
    string SystemCreationClassName;
    string SystemName;
    datetime TimeOfLastReset;

};

Methods

The Win32_NetworkAdapter class has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetPowerState</td>
<td>It defines the desired power state for a logical device and when a device should be put into that state. The desired power state is specified by setting the PowerState parameter to one of the following integer values: 1=&quot;Full Power&quot;, 2=&quot;Power Save - Low Power Mode&quot;, 3=&quot;Power Save - Standby&quot;, 4=&quot;Power Save - Other&quot;, 5=&quot;Power Cycle&quot; or 6=&quot;Power Off&quot;. The Time parameter indicates when the power state should be set, either as a regular date-time value or as an interval value (where the interval begins when the method invocation is received). When the PowerState parameter is equal to 5, &quot;Power Cycle&quot;, the Time parameter indicates when the device should power on again. Power off is immediate. SetPowerState should return 0 if successful, 1 if the specified PowerState and Time request is not supported, and some other value if any other error occurred. In a subclass, the set of possible return codes could be specified, using a ValueMap qualifier on the method.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Disable</td>
<td>It disables the network adapter.</td>
</tr>
<tr>
<td>Enable</td>
<td>It enables the network adapter.</td>
</tr>
<tr>
<td>Reset</td>
<td>It requests a reset of the logical device. The return value should be 0 if</td>
</tr>
<tr>
<td></td>
<td>the request was successfully executed, 1 if the request is not supported</td>
</tr>
<tr>
<td></td>
<td>and some other value if an error occurred.</td>
</tr>
</tbody>
</table>

**Table 4.7: Win32_NetworkAdapter Class Methods**

**Properties**

The Win32_NetworkAdapter class has the following properties.

- **AdapterType** - It reflects the network medium in use. This property may not be applicable to all types of network adapters listed within this class. Windows NT only.

  The network adapters are:

  - "Ethernet 802.3"
  - "Token Ring 802.5"
  - "Fiber Distributed Data Interface (FDDI)"
  - "Wide Area Network (WAN)"
  - "LocalTalk"
  - "Ethernet using DIX header format"
  - "ARCNET"
  - "ARCNET (878.2)"
  - "ATM"
  - "Wireless"
  - "Infrared Wireless"
  - "Bpc"
  - "CoWan"
  - "1394"

- **AdapterTypeID** - It reflects the network medium in use. This property gives the same information as the AdapterType property, except that the information is returned in
the form of an integer value that corresponds to the following: 0 - Ethernet 802.3 1 - Token Ring 802.5 2 - Fiber Distributed Data Interface (FDDI) 3 - Wide Area Network (WAN) 4 - LocalTalk 5 - Ethernet using DIX header format 6 - ARCNET 7 - ARCNET (878.2) 8 - ATM 9 - Wireless 10 - Infrared Wireless 11 - Bpc 12 - CoWan 13 - 1394 This property may not be applicable to all types of network adapters listed within this class.

Windows NT only.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0x0)</td>
<td>Ethernet 802.3</td>
</tr>
<tr>
<td>1 (0x1)</td>
<td>Token Ring 802.5</td>
</tr>
<tr>
<td>2 (0x2)</td>
<td>Fiber Distributed Data Interface (FDDI)</td>
</tr>
<tr>
<td>3 (0x3)</td>
<td>Wide Area Network (WAN)</td>
</tr>
<tr>
<td>4 (0x4)</td>
<td>LocalTalk</td>
</tr>
<tr>
<td>5 (0x5)</td>
<td>Ethernet using DIX header format</td>
</tr>
<tr>
<td>6 (0x6)</td>
<td>ARCNET</td>
</tr>
<tr>
<td>7 (0x7)</td>
<td>ARCNET (878.2)</td>
</tr>
<tr>
<td>8 (0x8)</td>
<td>ATM</td>
</tr>
<tr>
<td>9 (0x9)</td>
<td>Wireless</td>
</tr>
<tr>
<td>10 (0xA)</td>
<td>Infrared Wireless</td>
</tr>
<tr>
<td>11 (0xB)</td>
<td>Bpc</td>
</tr>
<tr>
<td>12 (0xC)</td>
<td>CoWan</td>
</tr>
<tr>
<td>13 (0xD)</td>
<td>1394</td>
</tr>
</tbody>
</table>

**Table 4.8: Win32_NetworkAdapter Types**

- **Availability**- The availability and status of the device. For example, the Availability property indicates that the device is running and has full power (value=3), or is in a warning (4), test (5), degraded (10) or power save state (values 13-15 and 17). Regarding the power saving states, these are defined as follows: Value 13 ("Power Save - Unknown") indicates that the device is known to be in a power save mode, but its exact
status in this mode is unknown; 14 ("Power Save - Low Power Mode") indicates that the device is in a power save state but still functioning, and may exhibit degraded performance; 15 ("Power Save - Standby") describes that the device is not functioning but could be brought to full power 'quickly'; and value 17 ("Power Save - Warning") indicates that the device is in a warning state, though also in a power save mode.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (0x1)</td>
<td>Other</td>
</tr>
<tr>
<td>2 (0x2)</td>
<td>Unknown</td>
</tr>
<tr>
<td>3 (0x3)</td>
<td>Running or Full Power</td>
</tr>
<tr>
<td>4 (0x4)</td>
<td>Warning</td>
</tr>
<tr>
<td>5 (0x5)</td>
<td>In Test</td>
</tr>
<tr>
<td>6 (0x6)</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>7 (0x7)</td>
<td>Power Off</td>
</tr>
<tr>
<td>8 (0x8)</td>
<td>Off Line</td>
</tr>
<tr>
<td>9 (0x9)</td>
<td>Off Duty</td>
</tr>
<tr>
<td>10 (0xA)</td>
<td>Degraded</td>
</tr>
<tr>
<td>11 (0xB)</td>
<td>Not Installed</td>
</tr>
<tr>
<td>12 (0xC)</td>
<td>Install Error</td>
</tr>
<tr>
<td>13 (0xD)</td>
<td>Power Save - Unknown</td>
</tr>
<tr>
<td></td>
<td>The device is known to be in a power save state, but its exact status is unknown.</td>
</tr>
<tr>
<td>14 (0xE)</td>
<td>Power Save - Low Power Mode</td>
</tr>
<tr>
<td></td>
<td>The device is in a power save state, but still functioning, and may exhibit degraded performance.</td>
</tr>
<tr>
<td>15 (0xF)</td>
<td>Power Save - Standby</td>
</tr>
<tr>
<td></td>
<td>The device is not functioning, but could be brought to full power quickly.</td>
</tr>
<tr>
<td>16 (0x10)</td>
<td>Power Cycle</td>
</tr>
<tr>
<td>17 (0x11)</td>
<td>Power Save - Warning</td>
</tr>
<tr>
<td></td>
<td>The device is in a warning state, though also in a power save state.</td>
</tr>
</tbody>
</table>

*Table 4.9: Win32_NetworkAdapter Availability*
• **DeviceID** - It contains a string uniquely identifying the network adapter from other devices on the system.

• **NetConnectionID** - Name of the network connection as it appears in the *Network Connections* Control Panel program.

• **NetConnectionStatus** - It specifies the name of the network connection as it appears in the 'Network Connections' folder.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0x0)</td>
<td>Disconnected</td>
</tr>
<tr>
<td>1 (0x1)</td>
<td>Connecting</td>
</tr>
<tr>
<td>2 (0x2)</td>
<td>Connected</td>
</tr>
<tr>
<td>3 (0x3)</td>
<td>Disconnecting</td>
</tr>
<tr>
<td>4 (0x4)</td>
<td>Hardware not present</td>
</tr>
<tr>
<td>5 (0x5)</td>
<td>Hardware disabled</td>
</tr>
<tr>
<td>6 (0x6)</td>
<td>Hardware malfunction</td>
</tr>
<tr>
<td>7 (0x7)</td>
<td>Media disconnected</td>
</tr>
<tr>
<td>8 (0x8)</td>
<td>Authenticating</td>
</tr>
<tr>
<td>9 (0x9)</td>
<td>Authentication succeeded</td>
</tr>
<tr>
<td>10 (0xA)</td>
<td>Authentication failed</td>
</tr>
<tr>
<td>11 (0xB)</td>
<td>Invalid address</td>
</tr>
<tr>
<td>12 (0xC)</td>
<td>Credentials required</td>
</tr>
</tbody>
</table>

*Table 4.10: Win32_NetworkAdapter Connection Status*

• **NetworkAddresses** - Array of network addresses for an adapter. This property is inherited from *CIM_NetworkAdapter*. This property has been implemented yet. It returns a NULL value by default.

• **PermanentAddress** - Network address hard-coded into an adapter. This hard-coded address may be changed by firmware upgrade or software configuration. If so, this field
should be updated when the change is made [100]. The property should be left blank if no hard-coded address exists for the network adapter. This property is inherited from CIM

- **NetworkAdapter.** This property has not been implemented yet. It returns a **NULL** value by default.

- **PhysicalAdapter**- Indicates whether the adapter is a physical or a logical adapter. If TRUE, the adapter is physical.

- **ProductName**- It indicates product name of the network adapter. Example: "Fast EtherLink XL"

- **ServiceName**- It indicates service name of the network adapter. This name is usually shorter than the full product name. Example: "Elnkii"

- **Status**- It is a string indicating the current status of the object. Various operational and non-operational statuses can be defined. Operational statuses are "OK", "Degraded" and "Pred Fail". "Pred Fail" indicates that an element may be functioning properly but predicting a failure in the near future. An example is a SMART-enabled hard drive. Non-operational statuses can also be specified. These are "Error", "Starting", "Stopping" and "Service". The latter, "Service", could apply during mirror-resilvering of a disk, reload of a user permissions list, or other administrative work. Not all such work is on-line, yet the managed element is neither "OK" nor in one of the other states.

The values are:

- "OK"
- "Error"
- "Degraded"
- "Unknown"
- "Pred Fail"
- "Starting"
• "Stopping"
• "Service"

• **StatusInfo** - It is a string indicating whether the logical device is in an enabled (value = 3), disabled (value = 4) or some other (1) or unknown (2) state. If this property does not apply to the logical device, the value, 5 ("Not Applicable"), should be used. This property is inherited from **CIM_LogicalDevice**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (0x1)</td>
<td>Other</td>
</tr>
<tr>
<td>2 (0x2)</td>
<td>Unknown</td>
</tr>
<tr>
<td>3 (0x3)</td>
<td>Enabled</td>
</tr>
<tr>
<td>4 (0x4)</td>
<td>Disabled</td>
</tr>
<tr>
<td>5 (0x5)</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>

**Table 4.11**: Win32_NetworkAdapter Status Information

• **SystemName** - Name of the scoping system. This property is inherited from CIM LogicalDevice.

### 4.1.4.5 Win32_NetworkAdapterConfiguration Class

The Win32_NetworkAdapterConfiguration WMI class represents the attributes and behaviors of a network adapter. This class includes extra properties and methods that support the management of the TCP/IP and Internetwork Packet Exchange (IPX) protocols that are independent from the network adapter [101]. The following syntax is simplified from Managed Object Format (MOF) code and includes all of the inherited properties. Properties are listed in alphabetic order, not MOF order.

**Syntax**

```csharp
class Win32_NetworkAdapterConfiguration : CIM_Setting
{
```
boolean  ArpAlwaysSourceRoute;
boolean  ArpUseEtherSNAP;
string   Caption;
string   DatabasePath;
boolean  DeadGWDetectEnabled;
string   DefaultIPGateway[];
uint8    DefaultTOS;
uint8    DefaultTTL;
string   Description;
boolean  DHCPEnabled;
datetime DHCPLeaseExpires;
datetime DHCPLeaseObtained;
string   DHCPServer;
string   DNSDomain;
string   DNSDomainSuffixSearchOrder[];
boolean  DNSEnabledForWINSResolution;
string   DNSHostName;
string   DNSServerSearchOrder[];
boolean  DomainDNSRegistrationEnabled;
uint32   ForwardBufferMemory;
boolean  FullDNSRegistrationEnabled;
uint16   GatewayCostMetric[];
uint8    IGMPLevel;
uint32   Index;
uint32   InterfaceIndex;
string   IPAddress[];
uint32   IPConnectionMetric;
boolean  IPEnabled;
boolean  IPFilterSecurityEnabled;
boolean  IPPortSecurityEnabled;
string   IPSecPermitIPProtocols[];
string   IPSecPermitTCPPorts[];
string IPSecPermitUDPPorts[];
string IPSubnet[];
boolean IPUseZeroBroadcast;
string IPXAddress;
boolean IPXEnabled;
uint32 IPXFrameType[];
uint32 IPXMediaType;
string IPXNetworkNumber[];
string IPXVirtualNetNumber;
uint32 KeepAliveInterval;
uint32 KeepAliveTime;
string MACAddress;
uint32 MTU;
uint32 NumForwardPackets;
boolean PMTUBHDetectEnabled;
boolean PMTUDiscoveryEnabled;
string ServiceName;
string SettingID;
uint32 TcpipNetbiosOptions;
uint32 TcpMaxConnectRetransmissions;
uint32 TcpMaxDataRetransmissions;
uint32 TcpNumConnections;
boolean TcpUseRFC1122UrgentPointer;
uint16 TcpWindowSize;
boolean WINSEnableLMHostsLookup;
string WINSHostLookupFile;
string WINSPrimaryServer;
string WINSScopeID;
string WINSSecondaryServer;
}
Methods

The Win32_NetworkAdapterConfiguration class has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisableIPSec</td>
<td>Disables IPsec on this TCP/IP-enabled network adapter.</td>
</tr>
<tr>
<td>EnableDHCP</td>
<td>Enables the Dynamic Host Configuration Protocol (DHCP) for service with this network adapter.</td>
</tr>
<tr>
<td>EnableDNS</td>
<td>Enables the Domain Name System (DNS) for service on this TCP/IP-bound network adapter.</td>
</tr>
<tr>
<td>EnableIPFilterSec</td>
<td>Enables IPsec globally across all IP-bound network adapters.</td>
</tr>
<tr>
<td>EnableIPSec</td>
<td>Enables IPsec on this specific TCP/IP-enabled network adapter.</td>
</tr>
<tr>
<td>EnableStatic</td>
<td>Enables static TCP/IP addressing for the target network adapter.</td>
</tr>
<tr>
<td>EnableWINS</td>
<td>Enables WINS settings specific to TCP/IP, but independent of the network adapter.</td>
</tr>
<tr>
<td>ReleaseDHCPLease</td>
<td>Releases the IP address bound to a specific DHCP-enabled network adapter.</td>
</tr>
<tr>
<td>ReleaseDHCPLeaseAll</td>
<td>Releases the IP addresses bound to all DHCP-enabled network adapters.</td>
</tr>
<tr>
<td>RenewDHCPLease</td>
<td>Renews the IP address on specific DHCP-enabled network adapters.</td>
</tr>
<tr>
<td>RenewDHCPLeaseAll</td>
<td>Renews the IP addresses on all DHCP-enabled network adapters.</td>
</tr>
<tr>
<td>SetArpAlwaysSourceRoute</td>
<td>Sets the transmission of ARP queries by the TCP/IP.</td>
</tr>
<tr>
<td>SetArpUseEtherSNAP</td>
<td>Enables Ethernet packets to use 802.3 SNAP encoding.</td>
</tr>
<tr>
<td>SetDatabasePath</td>
<td>Sets the path to the standard Internet database files.</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SetDeadGWDetect</td>
<td>Enables dead gateway detection.</td>
</tr>
<tr>
<td>SetDefaultTOS</td>
<td>Obsolete. This method sets the default Type of Service (TOS) value in the header of outgoing IP packets.</td>
</tr>
<tr>
<td>SetDefaultTTL</td>
<td>Sets the default Time to Live (TTL) value in the header of outgoing IP packets.</td>
</tr>
<tr>
<td>SetDNSDomain</td>
<td>Sets the DNS domain.</td>
</tr>
<tr>
<td>SetDNSServerSearchOrder</td>
<td>Sets the server search order as an array of elements.</td>
</tr>
<tr>
<td>SetDNSSuffixSearchOrder</td>
<td>Sets the suffix search order as an array of elements.</td>
</tr>
<tr>
<td>SetDynamicDNSRegistration</td>
<td>Indicates dynamic DNS registration of IP addresses for this IP-bound adapter. This method is new for Windows XP.</td>
</tr>
<tr>
<td>SetForwardBufferMemory</td>
<td>Specifies how much memory IP allocates to store packet data in the router packet queue.</td>
</tr>
<tr>
<td>SetGateways</td>
<td>Specifies a list of gateways for routing packets destined for a different subnet than the one this adapter is connected to.</td>
</tr>
<tr>
<td>SetIGMPLevel</td>
<td>Sets the extent to which the system supports IP multicasting and participates in the Internet Group Management Protocol.</td>
</tr>
<tr>
<td>SetIPConnectionMetric</td>
<td>Sets the routing metric associated with this IP-bound adapter.</td>
</tr>
<tr>
<td>SetIPUseZeroBroadcast</td>
<td>Sets IP zero broadcast usage.</td>
</tr>
<tr>
<td>SetIPXFrameTypeNetworkPairs</td>
<td>Starting with Windows Vista, the Internetwork Packet Exchange (IPX) technology is no longer supported and this method is obsolete. Windows Server 2003, Windows XP, and Windows 2000: Sets IPX network number or frame pairs for this network adapter.</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SetIPXVirtualNetworkNumber</td>
<td>Starting with Windows Vista, the Internetwork Packet Exchange (IPX) technology is no longer supported and this method is obsolete.</td>
</tr>
<tr>
<td>SetKeepAliveInterval</td>
<td>Sets the interval separating Keep Alive Retransmissions until a response is received.</td>
</tr>
<tr>
<td>SetKeepAliveTime</td>
<td>Sets how often TCP attempts to verify that an idle connection is still available by sending a Keep Alive packet.</td>
</tr>
<tr>
<td>SetMTU</td>
<td>Sets the default Maximum Transmission Unit (MTU) for a network interface. This method is not supported.</td>
</tr>
<tr>
<td>SetNumForwardPackets</td>
<td>Sets the number of IP packet headers allocated for the router packet queue.</td>
</tr>
<tr>
<td>SetPMTUBHDetect</td>
<td>Enables detection of Black Hole routers.</td>
</tr>
<tr>
<td>SetPMTUDiscovery</td>
<td>Enables Maximum Transmission Unit (MTU) discovery.</td>
</tr>
<tr>
<td>SetTcpipNetbios</td>
<td>Sets the default operation of NetBIOS over TCP/IP.</td>
</tr>
<tr>
<td>SetTcpMaxConnectRetransmissions</td>
<td>Sets the number of attempts TCP will retransmit a connect request before aborting.</td>
</tr>
<tr>
<td>SetTcpMaxDataRetransmissions</td>
<td>Sets the number of times TCP will retransmit an individual data segment before aborting the connection.</td>
</tr>
<tr>
<td>SetTcpNumConnections</td>
<td>Sets the maximum number of connections that TCP may have opens simultaneously.</td>
</tr>
<tr>
<td>SetTcpUseRFC1122UrgentPointer</td>
<td>Specifies whether TCP uses the RFC 1122 specification for urgent data, or the mode used by Berkeley Software Design (BSD) derived systems.</td>
</tr>
<tr>
<td>SetTcpWindowSize</td>
<td>Sets the maximum TCP Receive Window size offered by the system.</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>SetWINSServer</td>
<td>Sets the primary and secondary Windows Internet Naming Service (WINS) servers on this TCP/IP-bound network adapter.</td>
</tr>
</tbody>
</table>

**Table 4.12: Win32_NetworkAdapterConfiguration Class Methods**

**Properties**

The Win32_NetworkAdapterConfiguration class has the following properties.

**DefaultIPGateway**- It contains a list of IP addresses of default gateways used by the computer system. Example: “194.161.12.1 194.162.46.1”

**DHCPEnabled**- It indicates whether the dynamic host configuration protocol (DHCP) server automatically assigns an IP address to the computer system when establishing a network connection. Values: TRUE or FALSE. If TRUE, DHCP is enabled.

**DHCPServer**- It indicates the IP address of the dynamic host configuration protocol (DHCP) server. Example: 154.55.34

**DNSDomain**- It indicates an organization name followed by a period and an extension that indicates the type of organization, such as microsoft.com. The name can be any combination of the letters A through Z, the numerals 0 through 9, and the hyphen (-), plus the period (.) character used as a separator. Example: microsoft.com

**DNSDomainSuffixSearchOrder**- It specifies the DNS domain suffixes to be appended to the end of host names during name resolution. When attempting to resolve a fully qualified domain name (FQDN) from a host only name, the system will first append the local domain name. If this is not successful, the system will use the domain suffix list to create additional FQDNs in the order listed and query DNS servers for each. Example: samples.microsoft.com example.microsoft.com

92
**DNSEnabledForWINSResolution** - It indicates whether the Domain Name System (DNS) is enabled for name resolution over Windows Internet Naming Service (WINS) resolution. If the name cannot be resolved using DNS, the name request is forwarded to WINS for resolution.

**DNShostName** - It indicates the host name used to identify the local computer for authentication by some utilities. Other TCP/IP-based utilities can use this value to acquire the name of the local computer. Host names are stored on DNS servers in a table that maps names to IP addresses for use by DNS. The name can be any combination of the letters A through Z, the numerals 0 through 9, and the hyphen (-), plus the period (.) character used as a separator. By default, this value is the Microsoft networking computer name, but the network administrator can assign another host name without affecting the computer name. Example: corpdns

**IPAddress** - It contains a list of all of the IP addresses associated with the current network adapter. Example: 155.34.22.0

**IPConnectionMetric** - It indicates the cost of using the configured routes for this IP bound adapter and is the weighted value for those routes in the IP routing table. If there are multiple routes to a destination in the IP routing table, the route with the lowest metric is used. The default value is 1. Windows 2000 only.

**IPEnabled** - It indicates whether TCP/IP is bound and enabled on this network adapter. If **TRUE**, TCP/IP is bound and enabled on this network adapter.

**IPSubnet** - It contains a list of all the subnet masks associated with the current network adapter. Example: 255.255.0

**ServiceName** - The ServiceName property indicates the service name of the network adapter. This name is usually shorter than the full product name. Example: Elnkii.
4.1.5 Live-USB Drive Count Using WMI

Windows Management Instrumentation is a core Windows management technology that is used to get information about the internal state of computer systems, much like the disk drive information, network configurations, operating system detail etc. This system objects are modeled using classes such as Win32_LogicalDisk, Win32_DiskDrive, Win32_NetworkAdapter, Win32_NetworkAdapterConfiguration and Win32_OperatingSystem.

<table>
<thead>
<tr>
<th>S.NO</th>
<th>CLASS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Win32_LogicalDisk</td>
<td>represents the logical disks installed on a computer</td>
</tr>
<tr>
<td>2</td>
<td>Win32_DiskDrive</td>
<td>represents a physical disk drive as seen by a computer running the Win32 operating system</td>
</tr>
<tr>
<td>3</td>
<td>Win32_NetworkAdapter</td>
<td>represents a network adapter of a computer running a Windows operating system</td>
</tr>
<tr>
<td>4</td>
<td>Win32_NetworkAdapterConfiguration</td>
<td>represents the attributes and behaviors of a network adapters</td>
</tr>
<tr>
<td>5</td>
<td>Win32_OperatingSystem</td>
<td>returns an instance for the currently active operating system and list of operating system installed.</td>
</tr>
</tbody>
</table>

Table 4.13: Description of WMI Classes

With classes (1) and (2) in table 4.13, the disk drive counts of the system can be found. This helps to identify the Live-USB drive and boot the operating system. The boot configuration for windows XP and lower version of windows operating systems are same and for the higher version the boot manager is different. The equation to identify the Live-USB drive is

\[
\text{Live-USB} = \text{LogicalDisk} (\text{Type 5}) + \text{LogicalDisk} (\text{Type 2}) - 1
\]

Where \( \text{LogicalDisk} (\text{Type 5}), \text{LogicalDisk} (\text{Type 2}) \in \mathbb{N} \)

\[
0 < \text{LogicalDisk} (\text{Type 5}) < 3
\]

\[
0 < \text{LogicalDisk} (\text{Type 2}) \leq 127
\]

\[
1 < \text{LiveUSB} < 133
\]
LogicalDisk(Type 5) – Number of hard disk present in the system

LogicalDisk(Type 2) – Number of USB disk present in the system

The network information for the Live-USB can be fetched from the original system using the classes (3) and (4). Class (5) is used here to return current working operating system and list of operating system installed in hard disk and USB device.

The above said classes are included in the dynamic boot loader design. They serve their corresponding functionalities in the dynamic loader. The step by step development process of the dynamic boot loader is given in the following section.

4.2 DYNAMIC BOOT LOADER DESIGN

4.2.1 Introduction

A boot loader is the first software program that runs when a computer starts. It is responsible for loading and transferring control to operating system kernel. The kernel, in turn, initializes the rest of the operating system. It is a very powerful boot loader, which can load a wide variety of operating systems in USB drive, as well as proprietary operating systems with chain-loading [102]. It is designed to address the complexity of booting a personal Live-USB device; both the program and this manual are tightly bound to that computer platform, although porting to other platforms may be addressed in the future.

One of the important features in this boot loader is flexibility; it understands file systems and kernel executable formats, so you can load an arbitrary operating system the way you like, without recording the physical position of your kernel on the disk. Thus you can load the kernel just by specifying its file name and the drive and partition where the kernel resides. When booting with this loader the menu interface is used, and it is based on a configuration file which the loader prepares beforehand dynamically [103]. In the following chapters, you will discuss
how the loader specifies a drive, a partition, and a file name, how to configure network connectivity dynamically in Live-USB drive, and how to boot Live-USB.

### 4.2.2 Features of Boot Loader

This Boot Loader contains several features that make it preferable to other boot loaders available for the x86 architecture. Below is a partial list of some of the more important features:

- This loader provides a true command-based, post dynamic-OS environment on x86 machines. This feature affords the user maximum flexibility in loading operating systems with specified options or gathering information about the system. For years, many non-x86 architectures have employed post dynamic-OS environments that allow system booting from a command line.

- This Loader supports Logical Block Addressing (LBA) mode. LBA places the addressing conversion used to find files in the Live-USB drive or hard drive's firmware, and is used on many IDE and all SCSI hard devices. Before LBA, boot loaders could encounter the 1024-cylinder BIOS limitation, where the BIOS could not find a file after the 1024 cylinder head of the disk. LBA support allows the boot loader to boot operating systems from partitions beyond the 1024-cylinder limit, so long as the system BIOS supports LBA mode. Most modern BIOS revisions support LBA mode.

- This loader can read FAT, NTFS, ext2 and ext3 partitions. This functionality allows loader to access its configuration file, menu.lst, every time the system boots, eliminating the need for the user to write a new version of the first stage boot loader to the MBR when configuration changes are made [104].
4.2.3 Boot Loader Terminology

One of the most important things to understand before using this boot loader is how the program refers to devices, such as USB drives and partitions. This information is particularly important when compiling boot loader to boot Live-USB systems.

4.2.3.1 Device Names

When referring to a specific device with boot loader, do so using the following format (note that the parentheses and comma are very important syntactically):

\[(\text{<type-of-device>}, \text{<bios-device-number>}, \text{<partition-number>})\]

The <type-of-device> specifies the type of device from which boot loader boots. The two most common options are \texttt{hd} for a disk drive (USB disk, hard disk etc) or \texttt{fd} for a floppy disk. A lesser used device type is also available called \texttt{nd} for a network disk [105].

The <bios-device-number> is the BIOS device number. The primary IDE hard drive is numbered as 0, a secondary IDE hard drive is numbered as 1 and so on for other disk present in an system (Example: USB disk). This syntax is roughly equivalent to that used for devices by the kernel. For example, the a in hda for the kernel is analogous to the 0 in hd0 for dynamic boot loader, the b in hdb is analogous to the 1 in hd1, and so on.

The <partition-number> specifies the number of a partition on a device. Like the <bios-device-number>, all types of partitions are numbered starting at 0. However, partitions are specified using letters, with a corresponding to 0, b corresponding to 1, and so on.

To give an example, if a system has more than one USB drive/hard drive, boot loader refers to the first USB drive/hard drive as (hd0) and the second as (hd1). Likewise, boot loader refers to the first partition on the first drive as (hd0, 0) and the third partition on the second USB drive/hard drive as (hd1,2).
In general the following rules apply when naming devices and partitions under this boot loader:

- It does not matter if system hard drives are IDE or SCSI, all hard drives and all USB drive begin with the letters hd. The letters fd are used to specify floppy disk.
- To specify an entire device without respect to partitions, leave off the comma and the partition number. This is important when telling boot loader to configure the MBR for a particular disk. For example, (hd0) specifies the MBR on the first device and (hd3) specifies the MBR on the fourth device.
- If a system has multiple drive devices, it is very important to know how the drive boot order is set in the BIOS. But the dynamic boot loader ignores the BIOS boot priority settings and displays all installed operating systems both in USB drive and hard disk, it becomes critical and this loader boots the Live-USB first.

4.2.3.2 File Names and Block lists

When passing commands to loader that reference a file, such as a menu list, it is necessary to specify an absolute file path immediately after the device and partition numbers. The following illustrates the structure of such a command:

```
(<device-type><device-number>, <partition-number>)</path/to/file>
```

In this syntax, replace <device-type> with hd, fd, or nd. Replace <device-number> with the integer for the device. Replace </path/to/file> with an absolute path relative to the top-level of the device. It is also possible to specify files to loader that do not actually appear in the file system, such as a chain loader that appears in the first few blocks of a partition [106]. To load such files, provide a blocklist that specifies block by block where the file is located in the partition. Since a file is often comprised of several different sets of blocks, blocklists use a special syntax. Each block containing the file is specified by an offset number of blocks,
followed by the number of blocks from that offset point. Block offsets are listed sequentially in a comma-delimited list. The following is a sample blocklist:

$$0+50,100+25,200+1$$

This sample blocklist specifies a file that starts at the first block on the partition and uses blocks 0 through 49, 100 through 124, and 200. Knowing how to write blocklists is useful when using boot loader to load operating systems which require chain loading. It is possible to leave off the offset number of blocks if starting at block 0 [107]. As an example, the chain loading file in the first partition of the first hard drive would have the following name:

$$(hd0, 0)+1$$

The following shows the chainloader command with a similar blocklist designation at the loader command line after setting the correct device and partition as root:

chainloader +1

4.2.3.3. The Root File System and Boot Loader

The use of the term root file system has a different meaning in regard to boot loader. It is important to remember that loader's root file system has nothing to do with the operating system root file system [108]. The kernel command is executed with the location of the kernel file as an option. Once the kernel boots, it sets up the root file system that users are familiar with. The original loader root file system and its mounts are forgotten; they only existed to boot the kernel file.

4.2.4 Boot Loader Interfaces

This loader has three interfaces which provide different levels of functionality. Each interface allows users to boot the kernel of an operating system. The interfaces are as follows:
**Menu Interface**

This is the default interface shown when boot loader is configured by the execution program. A menu of operating systems or preconfigured kernels is displayed as a list, ordered by name [109]. Use the arrow keys to select an operating system or kernel version and press the enter key to boot it. If you do nothing on this screen, then after the time out period (3Min) expires boot loader will load the Live-USB.

**Menu Entry Editor Interface**

To access the menu entry editor, press the e key from the boot loader menu. The loader commands for that entry are displayed here, and users may alter these command lines before booting the operating system by adding a command line (o inserts a new line after the current line and O inserts a new line before it), editing one (e), or deleting one (d).

After all changes are made, the b key executes the commands and boots the operating system [110]. The Esc key discards any changes and reloads the standard menu interface. The c key loads the command line interface.

**Command Line Interface**

The command line interface is the most basic interface, but it is also the one that grants the most control. The command line makes it possible to type any relevant commands followed by the Enter key to execute them. This interface features some advanced shell-like features, including Tab key completion based on context, and Ctrl key combinations when typing commands, such as Ctrl-a to move to the beginning of a line and Ctrl-e to move to the end of a line. In addition, the arrow, Home, End, and Delete keys work as they do in the bash shell [111].

**4.2.4.1. Interfaces Load Order**

When the loader loads its second stage boot loader, it first searches for its configuration file. Once found, the menu interface bypass screen is displayed. Boot loader builds a menu list
and displays the menu interface. If no key is pressed, the default kernel present in the Live-USB is used.

4.2.5 Boot Loader Commands

This Boot Loader allows a number of useful commands in its command line interface. Some of the commands accept options after their name; these options should be separated from the command and other options on that line by space characters. The following table 4.14 is a list of useful commands used in this work.

<table>
<thead>
<tr>
<th>S.N O</th>
<th>COMMANDS</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>boot</td>
<td>Boots the operating system or chain loader that was last loaded.</td>
</tr>
<tr>
<td>2</td>
<td>chainloader</td>
<td>Loads the specified file as a chain loader. If the file is located on the first sector of the specified partition, use the blocklist notation, +1, instead of the file name. (e.g.) chainloader +1</td>
</tr>
<tr>
<td>3</td>
<td>initrd</td>
<td>Enables users to specify an initial RAM disk to use when booting. An initrd is necessary when the kernel needs certain modules in order to boot properly, such as when the root partition is formatted with the ext3, FAT or ext2 file system.</td>
</tr>
<tr>
<td>4</td>
<td>kernel</td>
<td>Specifies the kernel file to load when booting the operating system.</td>
</tr>
<tr>
<td>5</td>
<td>root</td>
<td>Configures the root partition for this loader, such as (hd1,0), and mounts the partition. (e.g.) root (hd1,0)</td>
</tr>
<tr>
<td>6</td>
<td>rootverify</td>
<td>Configures the root partition for this loader, just like the root command, but does not mount the partition.</td>
</tr>
</tbody>
</table>

Table 4.14: Boot Loader Commands

4.2.6. Boot Loader Menu Configuration

The configuration file menu.lst, which is used to create the list of operating systems to boot in menu interface, essentially allows the user to select a post-set group of commands to execute.
4.2.6.1. Configuration File Structure

This boot loader menu interface configuration file is menu.lst. The commands to set the global preferences for the menu interface are placed at the top of the file, followed by each operating kernel or operating system listed in the menu [112].

The following is an example of menu configuration file designed to boot a MS operating system present in Live-USB device and windows operating system present in hard disk.

```
default=0

Timeout=3

title MS Operating System
   root (hd1,0)
   kernel vmlinuz append root=/dev/ram0
   initrd /initrd.gz

# To load Windows

title Windows
   rootnoverify (hd0,0)
   chainloader +1
```

This file configures boot loader to build a menu with linux based MS operating system as the default operating system and sets it to auto boot after 3 seconds. Two sections are given, one for each operating system entry, with commands specific to the system disk partition table.

4.2.6.2. Configuration File Directives

The following are directives commonly used in this boot loader menu configuration file:

- `chainloader <path/to/file>` — Loads the specified file as a chain loader. Replace `<path/to/file>` with the absolute path to the chain loader. If the file is located on the first sector of the specified partition, use the blocklist notation, +1.
- `default=<integer>` — Replace `<integer>` with the default entry title number to be loaded if the menu interface times out.
- `initrd <path/to/initrd>` — Enables users to specify an initial RAM disk to use when booting. Replace `<path/to/initrd>` with the absolute path to the initial RAM disk.
- `kernel <path/to/kernel> <option-1> <option-N>` — Specifies the kernel file to load when booting the operating system. Replace `<path/to/kernel>` with an absolute path from the partition specified by the root directive. Multiple options can be passed to the kernel when it is loaded.
- `root (<device-type><device-number>,<partition>)` — Configures the root partition for boot loader, such as (hd0,0), and mounts the partition.
- `rootnoverify (<device-type><device-number>,<partition>)` — Configures the root partition for this boot loader, just like the root command, but does not mount the partition.
- `timeout=<integer>` — Specifies the interval, in seconds, that this boot loader waits before loading the entry designated in the default command.
- `title group-title` — Specifies a title to be used with a particular group of commands used to load a kernel or operating system.

### 4.2.7 Different Stages of Boot Loader

Boot Loader program loads the kernel of the operating system into the main memory for execution. The Boot Loader must be of size 512 bytes and should reside in the first sector of the disk drive. The procedure of Boot Loader is as follows:

Check the boot signature 0AA55h at 10,511-th bytes of the first sector of Boot Disk. If boot signature is present, it loads the code present in the first sector (512 bytes) to memory address 07c00h. Next the code at 07c00h is executed. This code then tries to find the available
physical memory and divides it into 64KB pages [113] [27] [32]. After that, 2 KB boot stack is allocated at (A0000-512) h and stack pointer is setup. Then Space for IVT (Interrupt Vector Table) and BIOS routines are reserved, and kernel is loaded at 00600h. The kernel that is to be loaded can be an EXE, BIN or COM file. Search for this kernel file will be done in the Root Directory (19th sector of the Boot Disk). On getting the file, it is allocated properly with all needed segments and memory pointers [13] [32]. If the kernel is in BIN or COM format it will have a single segment with all DS (Data Segment), CS (Code Segment, ES (Extra Segment), SS (Stack Segment) integrated. If the kernel is in EXE format, it will have separate code, data, extra and stack segments. In such cases the exe header will be ripped off and proper relocation factors are added as needed [114]. Virtually boot loader follows two stages of loader they are primary and secondary boot loaders. After this, the loaded kernel is executed.

4.2.7.1 Stage 1 Boot Loader

The primary boot loader that resides in the MBR (Master Boot Record) is a 512-byte image containing both program code and a small partition table [16]. The first 446 bytes are the primary boot loader, which contains both executable code and error message text. The next sixty-four bytes are the partition table, which contains a record for each of four partitions (sixteen bytes each) [115]. The MBR ends with two bytes that are defined as the magic number (0xAA55). The magic number serves as a validation check of the MBR. Figure 4.1 shows the logical sections of an MBR.

After BIOS finishes initializing platform hardware devices [116], it will load OSBMM (OSLoader Bootstrap Measurement Module) into host memory. And then the measurement code part of OSBMM will analyze the measurement information to check the integrity of MBR, Boot Sectors and OS Loaders. If the verification succeeds, OSBMM will load MBR into 0000H:
7C00H, and give controls to MBR to perform the normal bootstrap. If the verification fails, it will read backup from hard disk and recover the tampered files and code.

Normally the problems with the MBR of a system may prevent the system from booting. The MBR may be affected by malicious code, become corrupted by disk errors, or be overwritten by other boot loaders when experimenting with multiple operating systems on a host [14] [117]. This recipe describes one method of repairing the MBR using the recovery console. The master boot record can be taken as a backup and can be restore it later.

![Fig. 4.1: Logical Sections of an MBR](image)

The job of the primary boot loader is to find and load the secondary boot loader (stage 2). It does this by looking through the partition table for an active partition. When it finds an active
partition, it scans the remaining partitions in the table to ensure that they're all inactive [18]. When this is verified, the active partition's boot record is read from the device into RAM and executed.

**4.2.7.2 Stage 2 Boot Loader**

The second stage boot loader loads the compressed kernel image into memory and the kernel then setups the environment and starts to manage the resources. And then the boot loader places the appropriate root file-system image into memory. Once the kernel and the root file-system images are loaded into memory, the boot loader hand over the whole control of machine to the kernel [118] [31].

It does this by making the two-stage boot loader into a three-stage boot loader. Stage 1 (MBR) boots a stage 1.5 boot loader that understands the particular file system containing the Linux kernel image. Examples include reiserfs_stage1_5 (to load from a reiser journaling file system). When the stage 1.5 boot loader is loaded and running, the stage 2 boot loader can be loaded.

In stage 2, it will copy itself into SDRAM (Synchronous Dynamic Random Access Memory). In our implementation, after the initialization in stage 1, kernel of Live-USB drive will be copied from flash to offset 0x0000, 0000 of memory. This is the real boot program. It contains the user interface and the kernel loader [30].

With stage 2 loaded, it can display a list of available kernels defined in the boot menu. Normally you can select a kernel and even amend it with additional kernel parameters. Here by default it loads the live USB drive automatically. Optionally, with the end user interaction it will be able to boot the remaining kernal system [119]. With the second-stage boot loader in memory, the file system is consulted and the default kernel image and initrd image are loaded into memory. With these loaded images, the stage 2 boot loader invokes the kernel image.
4.2.8 Compile and Configure Boot Loader

The source code of the boot loader is made up of code editing in C language and assembly language, and these codes are compiled by Platform Builder. Build.exe need some file to determine what source code is need to be compile and how to compile and link the sources code, and the directories to traverse [120]. Therefore, programmer should edit the sources file; makefile file, Dirs file and Boot.bib file.

1) **Sources file:** A sources file is a text file that sets the macro definitions for the source code in a subdirectory. Build.exe uses these macro definitions to determine how to compile and link the source code [16].

2) **Dirs file:** The dirs file is a text file that specifies the subdirectories that contain source code to be built. The dirs files specify the sources files that contain source code to be built, and information on additional subdirectories that contain sources files. As the dirs file is not the importance of this section, it would not be detailed here [121].

3) **Makefile file:** Each subdirectory in a source code tree that includes a sources file also includes a makefile file, and in Platform Builder, the makefile file contains only the common makefile file, Makefile.def. Makefile.def contains some linking rules, using the rules, Nmake.exe compiles the source code specified in the sources files or it links object modules.

4) **Boot.bib:** As we know, the source code is compiled to be an .exe file, but the file that burn to flash is a binary image file, namely, the .bib file or .nb0 file. Using the boot.bib file, the eboot.exe is conversed to .bib file or .nb0 file. Eboot is built to run out of RAM, though initially the location of the image may be in flash. This .bib file will create two different Eboot images of interest [122].
After compiling, the dynamic boot loader is ready to use and it is in binary format. The objective of this boot loader is to load the Live-USB. For this purpose a light weight operating system called MS operating system is developed. The step by step process is discussed below.

4.3. OPERATING SYSTEM DESIGN

4.3.1. Introduction

MS operating system is a lightweight Linux distribution that focuses on use. The entire system can be run from RAM, allowing the boot medium to be removed after the operating system has started. The dynamic loader is embedded in a Mini MS Operating System and arrangements are made to select the required operating system by the user. The MS operating system is designed in such a way that inherits the properties of a base linux operating system. Applications such as AbiWord (a free word processing application), Gnumeric (a spreadsheet) and MPlayer (a free multimedia player) are included, along with a wide choice of web browsers that can be installed. This MS-OS build environment requires a base Linux Operating System with an ext3 file system and the necessary packages for the user convenience.

4.3.2. Setting the Build Environment

For creating the MS OS, We have to setup a build environment. First we need a separate partition for compilation process. So create minimum of 1.5GB partition with ext2 or ext3 file system and at least one Linux swap partition is must.

The existing puppy-unleashed core package is used (which provides all necessary scripts to extract the pet packages and to create the ISO files). Extract the puppy-unleashed package into the newly created partition [123]. puppy-unleashed folder will get while extracting the package. Inside the puppy-unleashed directory, it will have another one directory named packages. Copy
the required pet packages into the directory ‘packages’ [124]. These packages can also be obtained from website www.ibiblio.org.

4.3.3 Steps for Creating Build Environment

1. Creating a partition
   i. Boot any Puppy-linux from live-CD.(prefer Lucid puppy as it is easy to handle)
   ii. Create a partition with minimum 1.5GB using gparted partition manager (menu->system->gparted partition manager) with ext2 or ext3 file system.
   iii. The hard drive should also have a Linux swap partition, about 500M.
2. For mounting and getting into the partition (For example: /mnt/sda10), type the following command in terminal
   The created partition must be mounted in a folder to use it
   mkdir /mnt/MSOS (creates a folder MSOS in sda10)
   mount /mnt/sda10 /mnt/MSOS (Mounting sda10)
   cd /mnt/MSOS
3. Copy puppy-unleashed-core-2.17.tar.gz (I prefer this) into the newly created partition from our location.
4. Type the following command to extract the puppy-unleashed-core.tar.gz
   sync
   tar –zxvf puppy-unleashed-core.tar.gz
5. Copy all the packages from www.ibiblio.org into /mnt/MSOS/puppy-unleashed/packages directory. (These packages are used for creating new OS)
6. In the packages.txt file, each package has a separate line. To use a package on build, type “on” on particular package. If any extra packages have to be added which are not available
in the list, write the package in that file in the format as given in the file and set it “on”. If not needed, then set it “off”.

7. Type the following commands in terminal. It will extract all the pet packages in packages folder of MSOS-unleashed and it can be used them for further compilation.

   i. cd /mnt/MSOS/puppy-unleashed

   ii. sync

   iii. ./expandtarballs.sh

   The above command may take some time.

   During this command execution all the .pet packages in the puppy-unleashed/ packages are extracted to a folder. This area provides the way for customizing the newly developing operating system [125].

4.3.4. Customization

Customization in Drive Mount

Drive Mounter is used to mount all hard disk partitions, USB, CD etc. In puppy Linux, the mounter is actually a pet package called puppy drive mounter. It is the storage icon in the desktop of puppy linux [126].

Changes in MUT-29DEC06

The default drive mounter has a name Media Utility Tool (MUT). This name has to be changed as MS Drive Mounter. Then the status bar text and penguin icon in the MS Drive Mounter window has also to be changed.

1. Open the file **mut.tcl** from the location /mnt/MSOS/puppy-unleashed/packages/ mut-29dec06/ usr/lib/mut.

   mut.tcl is script which is programmed for setting the title name and the status bar text and place the penguin logo in the MS Drive Mounter window.
2. To change the title name of the MS Drive Mounter, Search for the line \texttt{wm title. "Media Utility Tool $::Version"} (Line number may be around 1191) in the file mut.tcl and replace with the following

\texttt{wm title . "MS Drive Mounter"}

3. To change the status bar text of the Drive Mounter, Search for the line \texttt{label .status\_bar -text "MUT $:: Version, written by Jesse Liley for Puppy Linux 2005."} (Line number may be around 1195) in the file mut.tcl and replace it with the following

\texttt{label .status\_bar -text "MS Operating System"}

4. To remove the penguin logo in the Drive Mounter window, Search for the following lines one by one and delete those lines (line numbers are around 221,222 and 1485 respectively)

\texttt{.boxc delete tux}

\texttt{.boxc create image $wx $wy -image ::img::img(tux) -anchor se -tag tux}

\texttt{image create photo ::img::img(tux) -file "$mut/icon/tux.ppm"}

To change the tux logos go to \texttt{usr/lib/mut/icon/tux.ppm} of mut-29dec06 directory inside packages. \texttt{tux.ppm} is the penguin logo. To replace this logo create a 48*48 portable pixmap image(.ppm) using any image editor and save it as \texttt{tux.ppm}

\textbf{4.3.5. Customization in File Manager}

A file manager is a computer program that provides a user interface to work with file systems. They are very useful for speeding up interaction with files. The most common operations on files are create, open, edit, view, print, play, rename, move, copy, delete, attributes, properties, search/find, and permissions [127]. In this OS, rox-filer is used as file manager.

\textbf{Setting Desktop Background}

To place the necessary icons (Home, Console, Exit, Write, and Browser) and set the MS Desktop background, the following steps should be followed.
1. Type the following commands in terminal to create a directory with the name backgrounds. The desktop image must be placed in to the background directory.

   `mkdir /mnt/MSOS/puppy-unleashed/packages /rox_filer-2.6.4.pup1/usr/share/backgrounds`

2. Copy the `default.jpg` to `/mnt/MSOS/puppy-unleashed/packages/rox_filer-2.6.4.pup1/usr/share/backgrounds`. `default.jpg` is the image which is going to be set as desktop background.

## Placing the Desktop Icons

**puppypin** is a xml file. It is used to place an application shortcut on desktop. If you want to provide any applications other than what we are having now, then put an entry in puppypin file. Open the **puppypin** file from `/mnt/MSOS/puppy-unleashed/packages/rox_filer-2.6.4.pup1/root/Choices/ROX-Filer/`.

   **Example:** `<icon x=”123” y=”789” label=”draw”>` places the draw application icon at x axis 123 and y axis 789.

## Changing the Icon of Rox

To change the default icon of the rox (Home) executable file. The following steps should be done.

   i. Copy the `img_home.gif` into the directory `/mnt/MSOS/puppy-unleashed/packages/rox_filer-2.6.4.pup1/usr/local/bin` from our location.

   ii. Right click on the file `rox` in the location `/mnt/MSOS/puppy-unleashed/packages/rox_filer-2.6.4.pup1/usr/local/bin` and select **File rox** from the menu, then select set Icon. It will open a window named **setIcon**.

   iii. Drag the copied `img_home.gif` icon to the **setIcon** window.
Changes in Desktop Icon Images (Globicons)

**Globicons** is a file which is in the Rox-Filer package (/mnt/MSOS/puppy-unleashed/packages/rox_filer-2.6.4.pup1/root/Choices/ROX-Filer/globicons). This file maintains the icons for the desktop applications (i.e. the path of the icon will be maintained in this file for all desktop applications) [128]. The procedure given below should be followed for setting the desktop icons.

1. Open globicons file from /mnt/MSOS/puppy-unleashed/packages/rox_filer-2.6.4.pup1/root/Choices/ROX-Filer/ globicons. This globicons is an xml file that links the corresponding icons from the desktop to its corresponding executables.

2. To change the desktop icons copy the images (.png 48*48) to /mnt/MSOS/puppy-unleashed/0rootfs_skeleton-2.17/usr/local/lib/X11/pixmaps, where all the images for desktop will be maintained.

Changing the Send to/Open with Option

This includes or removes the open with options in MS OS.

Within the path /mnt/sda1/MSOS/puppy-unleashed/packages/rox_filer2.6.4.pup1/root/.config/rox.sourceforge.net/openwith/ , the symlink of executables are included .

For eg: openwith-> vlc media player

Here the symlink added in the openwith is vlc media player.

Setting up a Boot Message

1. Open the boot.msg file from the location /mnt/sda1/MSOS/puppy-unleashed/isolinux-builds/

2. Delete all the contents from the boot.msg file and type as:

   SI0b MS OPERATING SYSTEM SI0a

   Here 0b and 0a are color codes……..

   In 0b 0- represents the background color
b-represents the foreground font color

Writing the Help Manual for MS OS

1. Open the folder from the path /mnt/sda1/MSOS/puppy-unleashed/
   0rootfs_skeleton/usr/share/doc

2. Edit the index-skeleton-top.htm file in html mode and add the MS OS details.

3. Delete all other files except wireless folder.

Changing the Puppy Country Prompt and Keyboard Prompt to MS Prompt

Search the puppy ‘country prompt’ and keyboard prompt from the location
/mnt/sda1/MSOS/puppy-unleashed/0rootfs_skeleton/etc/rc.d and change it as MS COUNTRY
PROMPT and MS KEYBOARD PROMPT.

4.3.6. Avoiding User Interaction While Booting

While booting, the user has been asked to enter the keyboard layout video wizard and
resolution. To avoid this complete user interaction, the following steps should be followed

Step 1: To Remove the Keyboard Layout Window

1. Keymap file should be copied to the location /mnt/MSOS/puppy-unleashed/packages
   /rox_filer-2.6.4.pup1/etc from our location.

2. The content of keymap file is us.map. us.map represents the US keyboard layout. For
   other keyboard layout the keymap file should be modified.

Step 2: To Remove the Video Mode (Resolution) Wizard

1. videomode file is copied to the location /mnt/MSOS/puppy-unleashed/packages/
   rox_filer-2.6.4.pup1 etc. from our location

2. In the videomode file, the resolution of Xvesa graphical server has been set as 0x118
   1024x768x24 which is maximum. The default resolution can be changed in this system.
Step 3: To Remove the Puppy Video Wizard at Startup

1. **xorgwizard** file is replaced into the folder `/mnt/MSOS/puppy-unleashed/packages/rootfs_skeleton-2.17/usr/sbin` from our location.

2. The modification is made in **xorgwizard** file to use only Xvesa graphical server. There are two graphical servers available namely Xorg and Xvesa. To use Xvesa, the resolution will be set for all systems which may differ from system to system [129]. But in the case of Xorg, the resolution will set for all systems which differ from the previous resolution.

4.3.7 Customization in Window Manager (ICEWM)

1. Create a directory with the name **root** in `/mnt/MSOS/puppy-unleashed/packages/icewm-1.2.31-i486` location by using the following command

   ```
   mkdir /mnt/MSOS/puppy-unleashed/packages/icewm-1.2.31-i486/root
   ```

2. Copy the **.icewm** folder from our location to the `/mnt/MSOS/puppy-unleashed/packages/icewm-1.2.31-i486/root`

3. Copy the **my-applications** folder in `/mnt/MSOS/puppy-unleashed/packages/icewm-1.2.31-i486/root`

4. Delete the directory **usr** in `/mnt/MSOS/puppy-unleashed/packages/icewm-1.2.31-i486/` location

5. The icewm file is compiled using the edited source package, which has been edited for disabling the right-click on taskbar.

6. Replace the **preferences**, **startup** and the **toolbar** files from our location into the `/mnt/MSOS/puppy-unleashed/packages/icewm-1.2.31-i486/root/.icewm` folder.

In the **preferences** file, modifications are made for the customization of taskbar like auto-hiding the taskbar, reducing the number of workspaces, removing the menu button from the taskbar, disabling the right-click on taskbar and removing the CPU-status from the taskbar [130].
**Toolbar** file has been modified for removing the quick launch button from taskbar and placing the EXIT icon on it.

**Startup** script has been changed to avoid the unnecessary package usage (for example mini-volume).

**Modifications in ICEWM Package**

1. The folders in /root/.icewm are copied from the puppy215 version. In this folder, the IceBergx folder is renamed as Icedesert in the themes folder of .icewm.

2. The following files are copied in /root/my-applications/bin location
   
   1. icewm-tray
   2. icesh
   3. icewmbg
   4. icewmhelp
   5. icewmhint
   6. icewm-session

**Change the Icons for Executable Files**

To Change the icon for MSConsole Executable File

1. Copy the img_MSconsole.png into the directory /mnt/MSOS/puppy-unleashed/packages/rxvt-2.6.4/usr/bin from our location.

2. Right click on the file rxvt file in the location /mnt/MSOS/puppy-unleashed/packages/rox_filer-2.6.4.pup1/usr/bin and select **File** option in **rxvt** menu, then select **setIcon** sub menu. It will open a window named **setIcon**

3. Drag the copied img_MSconsole.png into the setIcon window.

To Change the Icon for Default Texteditor Executable Files

1. Copy the img_edit.gif into /mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/usr/local/bin from our location.
2. Right click on the file defaulttexteditor file in the location /mnt/MSOS/puppy-unleashed/packages/rox_filer-2.6.4.pup1/usr/local/bin and select File defaulttexteditor menu, then select setIcon sub menu. It will open a window named setIcon.

3. Drag the above copied img_edit.gif into the setIcon window.

Now the customization of required packages has been completed. The next step is to create the ISO for MSOS which is described in the next chapter.

**To Set MS Browser**

Copy the MSbrowser script file from our location to /mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/usr/local/bin

To set firefox as MSbrowser, the content of MSbrowser script should be

```bash
#!/bin/sh
cd /opt/mozilla.org/bin
./firefox http://www.msuniv.edu.in
```

To set seamonkey as MSbrowser, write the following lines in MSbrowser script

```bash
#!/bin/sh
mozstart http://www.msuniv.edu.in
```

**4.3.8 Setup Booting and Shutting Down Screen**

1. Copy the bootsplash, pebble and pebble-daemon files into /mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/bin from our location. These three files are taken from the pebble package.

2. Copy the shutdown.jpg image into /mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/etc from our location.
4.3.9 Installing Packages

Installing Firefox

1. Download firefox-2.0.0.9.tar.gz and extract it in /mnt/MSOS/puppy-unleashed/packages/Mozill-firefox-2.0.0.9/usr/lib
2. Create a symlink for mozilla-firefox-2.0.0.9 by the name firefox.

Installing JRE

1. Download jre-6u1.tar.gz and extract it in /mnt/MSOS/puppy-unleashed/packages/jre-6u1/opt
2. Create a symlink for jre1.6.0_01 by the name jre.

Installing Flashfix

1. Download flashfix.pup and extract that package
2. Copy the files flashplayer.xpt and libflashplayer.so to root/.mozilla/plugins/

Issues in Firefox with Plugins

**Issue:**

1. The default page of firefox is http://www.msuniv.edu.in. It prompts to download the missing plug-ins.
2. The installation of those plug-in on clicking the link in that site leads the firefox to be closed. This will happen in all sites except www.msuniv.edu.in.

**Solution:**

1. In order to avoid this issue, we have to manually install the FlashFix package.
2. To do this in the ISO itself, copy the libflashplayer.so and its symlink into folder /root/.mozilla/plugins.
Issues while Detecting the ms_save.2fs from Different Partition

Issue:

Assume the situation that the puppy is booted from the CD and the pup_save.2fs file is in another drive (hard disk or flash drive). Then the pup_031.sfs and the zdrv_031.sfs files will be copied to the partition where the pup_save.2fs file is.

Solution:

This issue can be avoided by editing the init script in `/mnt/sda1/MSOS/puppy-unleashed/boot/init`. i.e by commenting out the line (usually line 377)

```
From

if [ “$PASSEDPARAM” = “save” ];then
#any *.sfs files in boot part. Should be copied to same part. as MS_save.2fs
fi

#printout to test
```

```
To

fi

#printout to test
```

4.3.10 Disable Single-Click Open for Files, Folders and Desktop Icons

Solution for Files and Folders

Edit the Options file in the path `/root/.config/rox.sourceforge.net/Rox-Filer` in RoxFiler package as

```
<Option name="bind_single_click">0</Option>
```

Which is originally

```
<Option name="bind_single_click">1</Option>
```

Solution for Desktop Icons

Edit the Options file in the path `/root/.config/rox.sourceforge.net/Rox-Filer` in RoxFiler package as
Which is originally

Which is originally

4.3.10.1 Disable Right-Click Open on Desktop Icons

Solution

1. Download the pup package Rox-2.5-NO-RCLICK.pup

2. Extract using the following command

   unzip Rox-2.5-NO-RCLICK.pup

3. There is a dotpup.tar.gz file in the extracted pup package. Then Extract the tar file and it gets the Rox-Filer executable.

4. In the Rox filer package, replace the Rox-Filer executable in the path /usr/local/apps/Rox-Filer with the above executable.

4.3.10.2 Changing Desktop Icon Text Color

Solution

Edit the Options File in the path /root/.config/rox.sourceforge.net/Rox-Filer in RoxFiler package as

Which is originally

Which is originally

4.3.10.3 Settings in Control Center

Puppy Background Picture Wizard

1. Add PuppyBackground package in packages.txt file

2. Open the PuppyBackground folder from the location /mnt/MSOS/puppy-unleashed/

3. Edit the help script in the puppyBackgroundpicture file as follows
cd /usr/lib/firefox-2.0.0.6
firefox file:///usr/local/PuppyBackground/resource/help.htm

4. Change the script name puppybackgroundmenu in /mnt/MSOS/puppy-unleashed/packages/PuppyBackground /usr/local/bin as msbackgroundmenu

5. Copy the background image files and paste those files in the location /mnt/MSOS/puppy-unleashed/packages/PuppyBackground/usr/share/backgrounds.

6. Add the line to the script “wmreboot” in the path /usr/X11R7/bin
   
   rm -rf /root/.pbthumbs

**Screensaver Settings**

1. Download xscreensaver.tar.gz and extract it in /mnt/MSOS/puppy-unleashed/packages/

2. Add xscreensaver package in packages.txt file

3. Add `xscreensaver-demo &` in /mnt/MSOS/puppy-unleashed/0rootfs_skeleton-2.17/root

   .xinitrc before the line `exec icewm`.

**Theme Settings**

1. The following files and folder has to be placed in the path /mnt/MSOS/puppy-unleashed/packages/icewm-1.2.31-i486 /root/.icewm
   - i. `linux` - This folder contains mime types for linux.
   - ii. `windows` - This folder contains mime types for windows
   - iii. `mstheme` – This folder contains mime types for mstheme.
   - iv. `themeslist` – This file contains list of themes in EKOS.

2. The following folders and files has to be placed in the path /mnt/MSOS/puppy-unleashed/icewm-1.2.31-i486 /root/.icewm/themes
   - i. `fedora`
   - ii. `xp`
iii. IcebergX

3. themeselection folder is placed in the path /usr/local

4. themeselection script is placed in the path /usr/bin

VPN Settings

1. VpnSettings.jar file is copied and paste in the path /root/my-appplication/ControlCtr

2. Swing_layout.jar file is copied and paste in the path /root/my-appplication/ControlCtr/lib.

4.3.10.4 Settings in Application Center

Setting Control Center and Application Center Icon in Desktop

1. Add the following lines to /mnt/MSOS/puppy-unleashed/rox-filer/choices/puppypin file:

   ```xml
   <icon x="704" y="96" label="Application Center">/usr/local/bin/Appcenter</icon>
   <icon x="576" y="96" label="Control Center">/usr/local/bin/Controlcenter</icon>
   ```

2. Add the following lines to globiicons file:

   ```xml
   <rule match="/usr/local/bin/ControlCenter">
     <icon> /root/my_applications/Controlctr/Icons/ControlSettings.png</icon>
   </rule>

   <rule match="/usr/local/bin/AppCenter">
     <icon> /root/my_applications/Controlctr/Icons/application.png</icon>
   </rule>
   ```

4. Appcenter and ControlCenter executable are placed in the path /mnt/MSOS/puppy-unleashed/packages/ControlCenter/usr/local/bin/

5. Icons for Settings and wizards in controlcenter and application center are placed in the location /mnt/MSOS/puppy-unleashed/ControlCenter/root/my-applications/Controlctr/Icons.
4.3.11 General Changes

1. **xwin** file in `/mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/usr/X11R7/bin` is replaced with our **xwin** file.

   If PC hang or doesn’t properly shutdown at the previous time, the question will be arisen to the user to type xwin while booting. To avoid this user interaction, the xwin has to be modified.

2. **hostname** file in `/mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/etc` is replaced with our **hostname** file from our location.

3. **puppy-reference** directory is deleted from the location `/mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/root`.

4. The contents of **puppyversion** file is deleted from `/mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/etc` and type 2012

   In the **MSOS version** file, our OS version has to be maintained.

5. Delete the content of the **OpenWith** directory from the location `/mnt/MSOS/puppy-unleashed/packages/rox_filer-2.6.4.pup1/root/.config/rox.sourceforge.net`

6. Replace the **rc.shutdown** file in `/mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/etc/rc.d` with our rc.shutdown script from our location. It will automatically create the ms_save.2fs file without the user interaction. In this file, command can also be used for setting shutting down screen.

7. Replace the **rc.sysinit** file in `/mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/etc/rc.d` with our rc.sysinit script from our location. This file is for boot screen (which is originally created by pebble package)

8. Replace the **profile** file in `/mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/etc` location with our profile script.

10. Copy the scripts `net-setup.sh`, `MSosinstaller`, `serialdetect` and `set-time` from our location to `/mnt/MSOS/puppy-unleashed/packages/0rootfs_skeleton-2.17/usr/sbin`.


12. The following packages, “puppybasic-1.1g (In order to run the pet and pup packages), ntfs_3g-20060714beta (To read and write into the ntfs partition) and perl-5.8.8-tiny (ndiswrapper depends on this package) are turned on.

13. Delete the following directories and files which are not needed for MS operating system:
   i. `/mnt/MSOS/puppy-unleashed/packages/xdg_puppy-0.7.6-3/etc/xdg/menus/*.menu` files
   ii. `/mnt/MSOS/puppy-unleashed/packages/xdg_puppy-0.7.6-3/usr/bin/jwm-xdgmenu`
   iii. `/mnt/MSOS/puppy-unleashed/packages/xdg_puppy-0.7.6-3/usr/bin/fvwm-xdgmenu`
   iv. `/mnt/MSOS/puppy-unleashed/packages/xdg_puppy-0.7.6-3/usr/share/Desktopdirectories`
   v. `/mnt/MSOS/puppy-unleashed/packages/rxvt-2.6.4/usr/share/applications`

   The following files are replaced with the files named wizard (MS OS) from our location:
   - bootmanager
   - burniso2cd
   - cdburnerwizard
   - cdburnerwizard-old
   - chooselocale
   - connectwizard
countrywizard
floppy-format.sh
inputwizard
modemwizard
MSosinstaller
pupdial
pupscan
printerwizard
petget
remasterpup2
resizepfile.sh
scannerwizard
timezone-set
video-wizard
wakeup2
wirelesswizard
wizardwizard
xf86vga336wizard
xorgwizard
xserverwizard

The following files are in the location /mnt/MSOS/puppy-unleashed/packages

bur2dvd2iso
complete2sfs
createpets
createpuppy
pinstall.sh

The file net-setup.sh is in the location /mnt/MSOS/puppy-unleashed/ and init is in

The script xinitrc in the locations /mnt/MSOS/puppy-unleashed/ boot /initrd-tree/
and /mnt/MSOS/puppy-unleashed/boot/initrd-tree/sbin are to be modified to avoid the
question to the user “which personal file you want to use”. This has been avoided by
mounting the ms_save.2fs file without any question to the user.
.xinitrc

15. To change the pup_ files in initrd, pup_031.sfs and pup_save.2fs, Change the word pup_ in
the following scripts as MS_

    init
    makeext2initrd
    createpuppy

The following scripts are in the location /mnt/MSOS/puppy-unleashed/boot/initrd-
tree and mnt/MSOS/ puppy-unleashed/

    bootmanager
    calcfreespace.sh
    delayedrun
    MSosinstaller
    petget
    partview
    remasterpup2
resizepfile.sh
savesession
savesession-dvd
snapmerge2flash
snapmerge2flash-aufsBROKEN
snapmerge2flash-unionfs
video-wizard

The following scripts are in the location /mnt/MSOS/puppy-unleashed/packages/0rootfs-skelton217/usr/sbin

pup2pet
dir2pet

The following scripts are in the location /mnt/MSOS/puppy-unleashed/packages/0rootfs-skelton217/usr/bin

modprobe
zdrvfind

The following script are in the location /mnt/MSOS/puppy-unleashed/packages/0rootfs-skelton217/sbin

xwin

The following script are in the location /mnt/MSOS/puppy-unleashed/packages/0rootfs-skelton217/usr/X11R7/bin

16. The name of the directories that starts with pup_ should be replaced with MS_ in the location /mnt/MSOS/puppy-unleashed/boot/initrd-tree

17. The word “puppy” has to be replaced in the file /mnt/MSOS/puppy-unleashed/isolinux-builds/boot.msg.
18. The word "default MSOS" and "label MSOS" in createpuppy script has to be replaced with "default MSOS" and "label MSOS"

19. The net_setup wizard package with the latest released has to be modified and the installer script has to be modified with the latest puppy linux version.

20. The warning message has to be changed that is displayed after booting MSOS.

   The following line in the file /usr/sbin/delayedrun has to be modified.

   yaf-splash -font “8x16” –outline 0 –margin 4 –bg yellow –text “warning! Do not unplug usb drive!” &

   as

   yaf-splash -font “8x16” –outline 0 –margin 4 –bg yellow –text “warning! Do not unplug MSOS drive!” &

21. To change desktop icon text color

   The following line in the file /root/.config/rox.sourceforge.net/Rox-filer/Options has to be modified.

   <option name=”pinboard_fg_color”>#000000000000<option>

   as

   <option name=”pinboard_fg_color”>#ffffffffffff<option>

4.3.12 Creating ISO for MS OS

   It is time to create the ISO file for our MS OS. To create ISO createpuppy script in the puppy unleashed package is used. Use the following commands in the terminal to run the createpuppy script [131]. While running the createpuppy script, it will prompt some questions, just hit Enter till you get the message ISO created. Press Ctrl-C keys for the next question. It will avoid writing the ISO file into the CD. Type the following command to execute the createpuppy script.
sync

cd /mnt/MSOS/puppy-unleashed/

./createpuppy

Now the complete ISO for MS Operating System is ready.

4.3.13 Booting From Live-CD

One thing to understand is that OS is unique. Barry built OS from scratch, file by file, so the architecture is not based on any other operating system distro. When the live-CD boots: vmlinuz, the Linux kernel, loads into RAM, followed by image.gz, which loads into a fast "ramdisk". The file is ".gz"-compressed, so it is uncompressed into the ramdisk and then the ramdisk becomes the very basic OS filesystem, that is, designated "/" and called top or root directory together with all subdirectories /bin, /sbin, /lib, /dev, /tmp, etc [132].

A fundamental objective of OS is that everything should load into ramdisk, thus freeing up the CD drive so that you can use the CD drive for other purposes eject. Also if everything is in RAM, application startup and running speed is stunning. Besides, a computer without spinning media (CD, hd) is much more rugged for use in boats etc.

When the power goes off, RAM disk content is lost. To retain our settings, such as configuration changes, email, browser history etc during boot up looks for a suitable flash drive partition, and if one is found, creates a file on it called "pup001". I arbitrarily chose "pupxxx" where the "xxx" is three numeric digits [133]. The three numeric digits have no particular meaning either, except we are currently using pup001 for the live-CD and pup100 for USB installations of OS. pup001 is a single file (easy to backup), but internally is a complete ext2 filesystem. After creating this file, OS mounts it, by means a "loopback device", onto /root directory. This directory /root is your home directory -- all your personal stuff goes in there. Thus, when you shutdown, all will still be there next time you start OS.I do need to explain the
bootup steps properly -- OS first has to mount the flash drive partition onto /mnt/home. This must be done before OS can get access to pup001 and mount it on root. So you can see two mount points in the figure 4.2.

Fig. 4.2: Live-CD Mount Points

Most of the files in OS, in any Linux distro for that matter, are in /usr – further more there is normally no need to write to anything inside /usr (that is, no need to edit, create or delete files). So the entire content of /usr is compressed as a single file, named usr_cram.fs and mounted as-is on /usr, by means of a "loopback device" -- the actual technique is not the main issue here, but you need to be aware that all of /usr is really only a single file, containing a compressed and read-only file system. Utilizing union-fs, however, changes in /usr can be made at run-time while usr_cram.fs remains unchanged [134]. Part of the figure 4.3 illustrates the different places where the file usr_cram.fs may be located. Those four numbers, 1, 2, 3, and 4, are the search order in case of booting off CD.

- At bootup, OS will first look to see if usr_cram.fs is at "/" in the ramdisk (which meant that it was inside image.gz originally)
- failing that, OS will look in /root (that is, inside pup001)
- else in /mnt/home (in the hard drive partition)
- finally OS will look on the CD (the slowest media of these)

Even if OS falls through to option 4, it will do his best not to leave usr_cram.fs on the slow CD. We really want to copy it to the ramdisk, or failing that to pup001 in the hard drive [135]. If we left it on the CD and just mounted it on /usr, OS will run slowly and the CD drive will not be available for other purposes. With enough RAM usr_cram.fs copies to "/" in the ramdisk, and then mounts it on /usr.

How much RAM is "enough"? OS is happiest on a PC with at least 128M RAM. 128M is a good size -- it is enough for image.gz (uncompressed) and usr_cram.fs to both reside on ramdisk.

What if your PC has only 64M, 48M, or even only 32M RAM? Well, usr_cram.fs is not going to fit in the ramdisk. Just ball-park figures 4.3, image.gz uncompressed is about 10M, usr_cram.fs is about 50M.

![Live CD Search Order](Fig. 4.3: Live-CD Search Order)
OS does have a card up his sleeve, so to speak -- if the PC has a swap partition OS will automatically use that, to increase the virtual size of the ramdisk. If you have ever installed another Linux distro on the PC, chances are it created a swap partition, so it is already sitting there ready for OS to use [136]. If your PC has 128M of RAM, OS will allocate 62M of that to ramdisk, however if the PC also has a 250M (for example) swap partition, then the effective size of the ramdisk becomes 62+250 = 312M!

However, I made a decision with the startup logic of OS, only to copy usrcram.fs into ramdisk if there is at least 62M of physical RAM allocated to the ramdisk. So, for PCs with less physical RAM than 128M, usrcram.fs will never be copied into ramdisk, even if there is a swap partition [137].

Another issue is the search order, those four numbers 1, 2, 3, and 4. When you have booted up OS from the CD, if you copy usrcram.fs from the CD or /, to /root (ie. inside pup001), that will be the second place OS looks next time he boots. Or, copy it onto "/" on the hard drive -- that's the 3rd place OS will look [138]. So even on a PC with only 32M RAM, you can boot up from a live-CD and have the CD drive freed up for other purposes.

PCs with less than 128M really do need a swap partition for OS to work properly. Even though OS can boot up from live-CD on a PC with only 32M RAM, some of the applications are memory hogs -- Mozilla for example. Mozilla is not viable on a PC with less than 128M RAM [139]. However, as mentioned above, a swap partition increases the effective size of the ramdisk, so you can get Mozilla to work on PCs with very little RAM, albeit slowly. It is also a good idea to add a swap file, if you have some spare space on the hard drive - and your partition tables allows it. Think about that search order a bit more... #3 is interesting. OS looks on the flash drive for usrcram.fs, but how would the file get there? You might have put it there manually or run the install-to-hard-drive-option-1 script.
However, when you boot a newer version of OS from CD, OS will find the old
usr_cram.fs! ...not so good. Or, an old usr_cram.fs if you put it in /root, for that matter [140].

To get around that problem OS has a technique for checking that usr_cram.fs is the
correct version. OS checks this, and will ignore any usr_cram.fs files it finds that have the wrong
version number. The technique used is when usr_cram.fs is created, its size, in bytes, is recorded
in /etc/sizeusrcram, which can then be used to check against usr_cram.fs at boot-time. With OS
current version it is often desirable to pursue the "everything on one piece of media" policy, but
you may split OS to overcome space limits on that media (CD-RW or USB stick) [141].

4.3.14 Booting From Live-USB

4.3.14.1 Live-USB System Architecture Overview

The way to understand the diagram is to view each of those layers as a complete file
system, that is, a complete directory hierarchy from root directory ("/") down. These layers are
laid one on top of the other, which is achieved by the unionfs file system. This file will be visible
at the top layer. If the "off-blue" layer has the same file, it will not be visible, as it is overlaid by
the same file on a higher layer [142]. Depending on this version of Linux you are running, the
method for creating the initial RAM disk can vary. The initrd is constructed using the loop
device. The loop device is a device driver that allows you to mount a file as a block device and
then interpret the file system it represents. The loop device may not be present in this kernel, but
you can enable it through the kernel's configuration tool by selecting Device Drivers > Block
Devices > Loopback Device Support. The small, but necessary, set of applications are present in
the. /bin directory, including nash (not a shell, a script interpreter), insmod for loading kernel
modules, and lvm (logical volume manager tools).
**Fig. 4.4:** Structure of MS Operating System

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ramdisk</td>
<td>This is the tmpfs filesystem running in RAM, with new and updated files.</td>
</tr>
<tr>
<td>ms_save.2fs</td>
<td>This is the persistent storage, where all your data, settings, email, installed packages, etc., get saved permanently. The &quot;.2fs&quot; means that the file contains a FAT filesystem.</td>
</tr>
<tr>
<td>ms_xxx.sfs</td>
<td>The built-in applications, window manager, scripts, everything. The &quot;.sfs&quot; means the file contains a squashfs compressed filesystem. The &quot;xxx&quot; is the version number without the dots.</td>
</tr>
<tr>
<td>*_xxx.sfs</td>
<td>These are additional squashfs files. The &quot;*&quot; can be anything. For example, devx_xxx.sfs is the complete environment for compiling C/C++ applications</td>
</tr>
</tbody>
</table>

While running this operating system, the outlook seen is one filesystem, which is the top layer. Thus you see `/usr/lib/libgdkxft.so` and you don't care what layer it is actually on. An exciting alternative to the squashfs extensions is to use an existing installed Linux distro as the bottom layer and it is shown in figure 4.5.
There are other variations, and it has a "state variable" named PUPMODE that shows what state, the operating system currently used [143]. There is a file, etc/rc.d/PUPSTATE that has the PUPMODE variable defined in the following modes:

- PUPMODE 5
- PUPMODE 12
- PUPMODE 13
- PUPMODE 2
- PUPMODE 77

**PUPMODE 5**

This is the configuration the very first time that operating system is booted from Live-USB and it is shown in figure 4.6. The first time that you plug in the USB and boot up, there is no persistent storage, and the "union" consists of only two layers, the top "working files" and the pup_xxx.sfs squashfs file system that has all the operating system files [144]. These two layers appear overlaid at root directory; however they can be viewed individually, at their respective mount points. So, we describe this approach but not touching the hard drive at all. You can run
applications, configure, download, install packages, but it is all happening in the tmpfs ramdisk, so not getting saved.

**Fig. 4.6:** First Time Boot Configuration of Live-USB

The way that we have been using pupmode is to create a "pup100" file on the USB drive, which has a FAT file system [145]. This file is copied into RAM at boot up, if there is enough RAM, thus avoiding writes to the Flash drive during a session. Then the files are copied back at shutdown.

The amount of space you have in the ramdisk depends on how much RAM is in the PC. The really interesting part is when you decide to end the session and shutdown the PC. The shutdown script, which is actually /etc/rc.d/rc.shutdown, will execute and will bring up a dialog window asking you to save the session with different allocation of memory size. Whatever directories and files that have been created in the ramdisk can now be saved [146]. The choice of storage location depends on whether a partition is a Linux filesystem, or FAT filesystem, a file called ms_save.2fs can be created in Live-USB drive and stored periodically every time.

**PUPMODE 13**

Configure an operating system to a USB Flash drive, perhaps by using the operating system Universal Installer program, you will have a bootable drive with the files vmlinuz (the
Linux kernel), initrd.gz (the initial ramdisk), ms_xxx.sfs (squashfs filesystem with all the os files) and syslinux.cfg (Syslinux config file). The situation is just like booting from a live-CD on first boot of this operating system and it will be in PUPMODE 5, as no persistent storage has yet been created [147]. On first shutdown, as described in the PUPMODE 5 section above, you will create a persistent storage called ms_save.2fs file. On the second boot, operating system will discover the persistent storage and boots. In the case of the persistent storage on Flash memory, which is the second layer and it is shown in figure 4.7. Operating system will save everything from the top layer to the second layer every 30 minutes [148].

Fig. 4.7: Second Time Boot Configuration of Live-USB

From the "unionfs" point of view, the second layer is mounted read-only, it is only the top layer that is written to, however this design can able to "flush" the top layer down to the next layer at periodic intervals [149]. This has an option to select the user storage in and operating system, according to this user can use two storage one in operating system and another in normal disk drive. The updating made in the operating system is stored in that particular space allotted for it.
Packaging

Once it has the kernel with all required functionalities ready, it requires writing it into a medium like USB disk drives. Remember that the boot loader always loads the executable in one specific sector of the drive. So it is important that to place the program in correct sector for the boot loader to find it and load it into memory [150]. Tools like masm and debug allows writing the executable program to specific sectors that specify.

Package installation

A key point about this architecture is that /usr is read-only. After downloading a .tar.gz "tarball" or RPM package you cannot install it into OS because most packages want to install part of themselves into /usr.

However, version 1.0.1 has revolutionized package management in OS by using UNIONFS to make /usr read-write. Any files written to /usr are actually stored in /root/.usr, but look as if they are in /usr to the RPM packs and such. A package manager with non-volatile installation, removal and dependency checking named PupGet is feasible thus!

PupGet offers the packages of the OS Unleashed package suite. All of these packages are available for download -- from the OS menu choose "Setup -> PupGet package manager", and follow the steps in the GUI.

The unleashed packages are all on the Internet, and PupGet can download them individually to install them, however Linux developers can create their own Live-USB from the complete unleashed package suite [151]. The live-CD that you are using now was created from OS Unleashed. OS Unleashed enables you to put together your own custom OS boot USB from a choice of hundreds of packages (applications).
4.3.14.2 Flash Technology

A 128MB Flash drive matches perfectly with OS. Consider, OS is only about 80-90MB, so will fit on a 128M USB drive leaving over 70MB free for personal data. Yes, a complete portable operating system and personal data, all on the one tiny non-spinning noiseless device [152]. The downside to flash technology is limited number of write-cycles. After so many times, the stick will collapse. I have seen figures as low as 50,000 writes. Anyway, the good news is that OS is especially designed to have no writes to the Flash drive during a session, enormously extending its life span. Take a look at this figure 4.8

![Diagram of OS boot from Live-USB](image)

**Fig. 4.8:** Layout of Operating System at Second and Later Boot ups from a Live-USB

When OS boots from Live-USB, the steps are much the same as for the live-CD. The kernel vmlinuz is loaded into RAM, image.gz is uncompressed and loaded into a ramdisk. OS boot from Live-USB. OS then searches for usr_cram.fs, first looking in "/" in the ramdisk, then in "/" in the USB partition. The first case would only be if OS had been created with
usr_cram.fs inside image.gz, which is not the normal situation. So it falls back to number 2. OS will find usr_cram.fs in the USB partition and will mount it on /usr.

OS will attempt an optimisation. If there is enough RAM, OS will copy usr_cram.fs into the ramdisk and then mount it on /usr. This takes a little while but improves running speed. However, even if OS does leave usr_cram.fs on the Live-USB drive and mounts it from there onto /usr, that is not a problem as /usr is read-only [153]. There will be no writes to /usr, so the lifetime of the Flash drive is not compromised. The UNIONFS which makes /usr read-write, but all writes actually go to /root/.usr, which is in the ramdisk, as described below.

The very first time that OS boots from Live-USB, the file pup100 does not yet exist -- OS has to create it. OS creates it, and mounts it directly on /root - physically on the stick - by means of a loopback device. So far, the architecture is basically the same as for the live-CD [154].

The lifetime of the Flash device is compromised by writes to /roots, that is, to pup100. This is your home directory, and applications store all kinds of stuff in it. A lot of writing will be happening to /root. The Linux kernel does cache the writes, but even so, the cache will be flushed frequently. A significant difference occurs the second time that OS is booted from Live-USB. If there is enough RAM (remembering that the ramdisk can also use a swap partition if it exists, so the available space is effectively the size of RAM plus size of the swap partition), then OS will mount pup100 onto /mnt/pupxxx directory in order to copy all the files from /mnt/pupxxx to /root. Thus, a complete copy of pup100 is in /root, and no writing will occur to physical pup100 during a session. At shutdown, all the files in /root are copied back to /mnt/pupxxx, thus updating pup100. We could, in theory, backup to pup100 at regular intervals, rather than only at shutdown, but that is not implemented in the current version of OS -- quite frankly, I have very rarely had OS crash, and the only situation that would cause backup not to occur would be a power failure.
OS is highly intelligent, and the optimisations described above are only implemented if there is enough RAM (+swap). If there is not enough RAM, then obviously pup100 cannot be copied into /root in the ramdisk. During boot OS tells you whether it manages to get all into RAM. To give an idea of what is "enough" RAM, a PC with 256M RAM matches well with a 128M Flash drive. The pup100 file will be about 60M, and there's enough RAM for everything to load into ramdisk. On the other hand, a 256M Flash drive would have a 180M pup100 file which would be too big. With a 256 flash-drive 512MB RAM are good to keep all in RAM. If the PC has a swap partition, OS will automatically use this to increase the effective maximum size of the ramdisk. Whatever the situation on your PC, OS will work out the best configuration, totally automatically.

As you can see, OS consists of only 4 files. (Actually, it is possible to have usr_cram.fs inside image.gz, reducing OS down to just 3 files -- the web download sites may have that configuration available. It is convenient for booting off a network to have just the two files vmlinuz and image.gz to worry about)

To update to the latest version of OS, you just need to get the latest vmlinuz, image.gz and usr_cram.fs. Updating is so very simple, but with v0.9.8 I made it even simpler by building an update option into the installation scripts. That is, boot up the latest live-CD and run the install-to-USB, install-to-Zip or install-to-hard-drive script and choose "update" rather than "new installation".

OS does have a couple of archive programs, including one that I wrote that backs up only the changed files from /root each time. However, all your personal data, really, a snapshot of the entire the state of the system, is in the pup001 or pup100 file, so you can save a complete system snapshot just by making a copy of the pup001/100 file [155]. Then compress it, like this (assuming the copy is named pup100-12dec04):
Gzip pup100-12dec04

and you will have pup100-12dec04.gz. Archive that anywhere you want.

You can mount an archived pup100 file at any time, and view its contents:

losetup-full /dev/loop2 pup100-12dec04

mount -t ext2 /dev/loop2 /mnt/data

You can also view the contents of pup100-12dec04 from Windows, by using the Explore2fs program. Having many pup100 e.g. on a DVD-RW makes sense to fully utilize all DVD space while maintaining the "1 media policy" and full "all-in-RAM speed" by keeping a moderately sized individual pup100. From the menu, pup001 on hd can be enlarged (you may know this word from your spam mail) [156].

4.3.14.3 Booting Operating System

Step 1

The first things that happen is the Linux kernel in file "vmlinuz" loads into RAM and file "image.gz" is uncompressed and loaded into /dev/ram0, a 11'264 K fixed-size ramdisk.

In the case of a "big image" OS, that is, with file usr_cram.fs inside image.gz, there has to be the kernel boot parameter "ramdisk_size=" to specify a suitable size to hold all of image.gz (uncompressed) and usr_cram.fs plus some spare space. For example:

ramdisk_size=63488

For the "big image" situation, /dev/ram0 will be large, in the above example 63488 Kbytes.

Step 2

The program "/sbin/init" is executed. In the case of the "big image" OS, /sbin/init is just a link to /bin/busybox, as init is a program built-in to busybox. The normal situation though, is that /sbin/init is a shell script. Here it is:
#!/bin/sh

#v0.9.8 booting up in /dev/ram0, move root / to tmpfs...
INITARGS=""'echo -n "$@" | sed -e '/\dev/\ram0/tmpfs/g'"'"PATH=/bin:/sbin

mount -t proc none /proc

mount -o remount,rw / #-n option not needed with busybox mount.
STARTUPFSSIZE=`cat /root0/.etc/ramdiskfssize` #currently set to 11264K (file gets updated below)

#specifying ram1 here as it isn't in use...
SIZERAMDISKM=`disktype /dev/ram1 | grep "Block device" | cut -f 4 -d " "`
SIZERAMDISKK=`expr $SIZERAMDISKM \* 1024`

#total ram, less any shared video...
PCRAMSIZE=`free | head -n 2 | tail -n 1 | tr -s " " | cut -f 3 -d " "`

#...note, busybox free output format different from standard free program.
if [ $PCRAMSIZE -gt 220000 ];then #256M
  SIZEFILLK=`expr $PCRAMSIZE \/ 2` #half of ram.
else
  if [ $PCRAMSIZE -gt 117000 ];then #128M
    SIZEFILLK=67584
  else
    if [ $PCRAMSIZE -gt 60000 ];then #64M
      SIZEFILLK=16384 #leaves about 7M free in f.s.
    else
      SIZEFILLK=11264 #only about 2M free in f.s.
    fi
  fi
fi
```bash
fi
fi

# precaution...
if [ $SIZEFILLK -lt $STARTUPFSSIZE ];then
    SIZEFILLK=$STARTUPFSSIZE
    Fi
    PHYSICALFILLK="$SIZEFILLK"
# want to know if there is a swap partition available...
    SWAPPART=  
    SWAPINFO="`fdisk -l | grep "Linux swap" | head -n 1`" if [ ! "$SWAPINFO" ];then
    # we can make the ramdisk real big now...
    SWAPPART="`echo "$SWAPINFO" | cut -f 1 -d " "`"
    SWAPSIZE="`fdisk -s $SWAPPART`"
    #SWAPSIZE=`expr $SWAPSIZE / 2` # let's use half of the swap partition.
    #... nah, grab the whole lot. OS can create a swap file later if reqd.
    SIZEFILLK=`expr $SIZEFILLK + $SWAPSIZE`
    # 0.9.8... but have to delay swapon until rc.sysinit running... see below...
    fi
mkdir /new_root
mount tmpfs /new_root -t tmpfs -o size=${SIZEFILLK}k;check_status $?
cp -a /bin /new_root/
cp -a /dev /new_root/
cp -a /etc /new_root/
cp -a /lib /new_root/
```
cp -a /mnt /new_root/
cp -a /root /new_root/
cp -a /root0 /new_root/
cp -a /sbin /new_root/
cp -a /usr /new_root/
cp -a /proc0 /new_root/proc
cp -a /var0 /new_root/var
mkdir /new_root/tmp
chmod 777 /new_root/tmp
sync

#0.9.8R2 rc.sysinit will start the swap partition running...
if [ ! "$SWAPPART" = "" ];then
echo -n "$SWAPPART" > /new_root/root0/.etc/swappartition1
fi

#i want to know the max size if not going to use the swap partition...
echo -n "$PHYSICALFILLK" > /new_root/root0/.etc/ramdiskphysicalfssize
echo -n "$SIZERAMDISKK" > /new_root/root0/.etc/ramdisksize
echo -n "$SIZEFILLK" > /new_root/root0/.etc/ramdiskfssize #size of f.s.
cat /etc/fstab | sed -e 's/\dev\/ram0/tmpfs/g' > /new_root/etc/fstab

#if ever run install-to-hd script, /sbin/init must be normal symlink...
mv -f /new_root/sbin/init /new_root/sbin/init_ORIG
ln -s /bin/busybox /new_root/sbin/init

cd /new_root
mkdir old_root
umount /proc

# have code in /etc/rc.d/rc.sysinit which will umount /dev/ram0...

pivot_root . old_root

exec chroot . sh -c "exec /bin/busybox init $INITARGS" dev/console 2>&1

In a nutshell, what this script does is copy all of /dev/ram0 to a new tmpfs ramdisk, which has the advantage of variable size and it can use a swap partition to create a very large effective size for the ramdisk. After the swap, /dev/ram0 is no longer being used.

Notice the very last line of the script. After the swap-over (chroot), "/bin/busybox init" is executed. This causes the init program built into Busybox to execute, the same as if /sbin/init was a link to /bin/busybox and init had been executed.

### 4.3.14.4 Temporary File System

The tmpfs is one of the ramdisk technologies in Linux. A RAM disk is a portion of RAM which is being used as if it were a disk drive. Access time is much faster for a RAM disk than for a real physical disk. Thus, placing the files into memory will increase the performance of computer. The tmpfs distinguishes itself from the Linux ramdisk device by allocating memory dynamically and by allowing less-used pages to be moved onto swap space [157]. These characteristics make tmpfs more flexible than ramfs and some older versions of ramfs. In this study, tmpfs is adopted to create a RAM disk for storing the memory running system during system boot process. Temporary file system is supported by the Linux kernel from version 2.4 and this system has been implemented using the kernel version 2.6.

### 4.3.14.5 The Kernel

Kernel is nothing but a program that resides in the memory, takes in user inputs, process those user inputs and give out a suitable response. Kernel can be EXE, BIN or COM file. Though the tasks done by the kernel varies with design, and this operating systems delegate system
functions to different layers or child programs, the absolute minimal functions common to all other sub-systems should be kept in the kernel [158].

When the kernel is loaded, it immediately initializes and configures the computer's memory and configures the various hardware attached to the system, including all processors, I/O subsystems, and storage devices. It then looks for the compressed initramfs images in a predetermined location in memory, decompresses it directly to /sysroot/, and loads all necessary drivers [159]. Next, it initializes virtual devices related to the file system, before completing the initramfs processes and freeing up all the memory the disk image once occupied.

At this point, the kernel is loaded into memory and operational. However, since there are no user applications that allow meaningful input to the system, not much can be done with the system.

To set up the user environment, the kernel executes the /sbin/init program. The /sbin/init program (also called init) coordinates the rest of the boot process and configures the environment for the user.

When the init command starts, it becomes the parent or grandparent of all of the processes that start up automatically on the system. First, it runs the /etc/rc.d/rc.sysinit script, which sets the environment path, starts swap, checks the file systems, and executes all other steps required for system initialization [160]. For example, most systems use a clock, so rc.sysinit reads the /etc/sysconfig/clock configuration file to initialize the hardware clock. Another example is if there are special serial port processes which must be initialized, rc.sysinit executes the /etc/rc.serial file.

Next, the init command sets the source function library, /etc/rc.d/init.d/functions, for the system, which configures how to start, kill, and determine the PID of a program. The init program starts all of the background processes by looking in the appropriate rc directory for the
runlevel specified as the default in /etc/inittab. The rc directories are numbered to correspond to the runlevel they represent. For instance, /etc/rc.d/rc5.d/ is the directory for runlevel 5. When booting to runlevel 5, the init program looks in the /etc/rc.d/rc5.d/ directory to determine which processes to start and stop [161].

4.3.14.6 Initial Ramdisk

The initial ramdisk is a temporary file system commonly used in the boot process of the Linux based kernel [162]. It is typically used for making preparations before the real root file system can be mounted. Initrd provides the capability to load a RAM disk by the boot loader [163].

![Diagram of boot process](image)

**Fig. 4.9:** Process Involved in Booting and Configuring Network in Live-USB

This RAM disk can then be mounted as the root file system and programs can be run from it. Afterwards, a new root file system can be mounted from a different device. The previous root (from initrd) is then moved to a directory and can be subsequently unmounted [164] [165]. Linux based operating system is booted from a Live-USB drive and initrd makes preparations for
shifting to real root file system which resides in a Live-USB drive. Normally kernel loads with their modules; along with this the network data that fetched from the original system is configured in Live USB system which is shown in figure 4.9.

Now the complete MS operating system is ready and it is integrated with dynamic boot loader as Live-USB. In the next chapter we discuss the step by step experimental result and the performance of dynamic boot loader and MS operating system.