**📘 Complete Theory Notes DWS Batch 4 📘**

**Date 7/8/25**

**1. Appwiz.cpl (Programs and Features)**

* Shortcut to Programs and Features window in Control Panel.
* Purpose: To view, uninstall, change, or repair installed applications on a system.
* Can be opened by:
  + Run dialog → appwiz.cpl
  + Command Prompt → control appwiz.cpl

**📍 Where the list comes from**

* **Reads from Registry keys:**
  + HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall
  + HKLM\Software\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall (for 32-bit apps)
  + HKCU\Software\Microsoft\Windows\CurrentVersion\Uninstall (user-specific apps)

**📍 Important details stored in registry**

* DisplayName → Name shown in the list
* DisplayVersion → Version of app
* Publisher → Vendor name
* InstallDate → Date of installation
* UninstallString → Command to uninstall app
* QuietUninstallString → For silent uninstallation (no prompts)

**📍 Usage**

* Uninstall/Repair software from GUI.
* Check installed updates or remove updates.
* Enable/Disable Windows features (optionalfeatures.exe).

**📍 Commands**

* Uninstall MSI package:
* msiexec /x {ProductCode} /qn
* List installed apps (PowerShell):
* Get-ItemProperty HKLM:\Software\Microsoft\Windows\CurrentVersion\Uninstall\\* | Select DisplayName, DisplayVersion

**2. Event Viewer**

* Tool for viewing system, application, and security logs.
* Helps in troubleshooting, monitoring, and auditing.
* **Open by:**
  + Run → eventvwr.msc
  + Search: “Event Viewer”

**📍 Main Log Categories**

1. Application → Logs from applications (errors, crashes).
2. System → Logs from OS, drivers, and services.
3. Security → Login attempts, account changes, auditing.
4. Setup → Installation and update logs.
5. Forwarded Events → Logs collected from other systems**.**

**📍 Important Event IDs**

* 41 → Kernel-Power (unexpected shutdown).
* 4624 → Successful logon.
* 4625 → Failed logon.
* 4720 → User account created.
* 4732 → User added to a group.
* 7031/7036 → Service stopped/started.
* 1000 → Application crash.

**📍 Uses**

* Troubleshoot system crashes and application errors.
* Check service failures and restarts.
* Track unauthorized login attempts or account changes.
* Monitor system performance and health.

**3. Services (services.msc)**

* Background processes that run independently of user logins.
* Managed by Service Control Manager (SCM).
* **Open:**
  + Run → services.msc
  + PowerShell → Get-Service

**📍 Service Properties**

* **Name & Description** → Identification**.**
* **Startup type:**
  + Automatic → Starts at boot
  + Automatic (Delayed Start) → Starts after boot
  + Manual → Starts only when needed
  + Disabled → Cannot be started
* Log On As → Defines which account the service uses (Local System, Network Service, etc.).
* Dependencies → Other services that must run first.
* Recovery options → Actions if service fails (restart, run program, reboot system).

**📍 Common Examples**

* wuauserv → Windows Update.
* spooler → Printer Spooler.
* BITS → Background Intelligent Transfer Service.
* Dnscache → DNS Client.

**📍 Commands**

* Check services (PowerShell):
* Get-Service
* Start/Stop service:
* Start-Service spooler
* Stop-Service spooler

**4. Environment Variables**

* Key-value pairs that affect how processes run on Windows.
* Two types:
  + User variables → Specific to logged-in user.
  + System variables → Global for all users.

**📍 Common Environment Variables**

* PATH → Directories searched for executables.
* TEMP/TMP → Temporary file storage.
* SystemRoot → Windows folder (usually C:\Windows).
* USERNAME → Current logged-in user.
* COMPUTERNAME → System’s name.
* ProgramFiles → Location of installed programs.

**📍 Where stored**

* System variables → HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment
* User variables → HKCU\Environment

**📍 Usage**

* Configure application behavior (e.g., JAVA\_HOME, NODE\_PATH).
* Modify PATH to run programs from any directory.
* Control temp storage, proxy settings, etc.

**📍 Commands**

* View variable:
* echo %PATH%
* Set variable (current session):
* set VAR=value
* Permanent change (PowerShell):

[Environment]::SetEnvironmentVariable("JAVA\_HOME", "C:\Java\jdk", "Machine")

**08/08/2025**

**1. Device Driver and its Types**

**🔹 What is a Device Driver?**

* A **device driver** is a special type of software that allows the **operating system (OS)** and **applications** to communicate with hardware devices.
* Without drivers, hardware like printers, keyboards, network cards, or graphic cards cannot work properly, because the OS doesn’t directly understand hardware commands.

👉 Think of a driver as a **translator** between the OS and the device.

**🔹 Functions of a Driver**

* Tells OS how to use hardware.
* Handles input/output (I/O) requests.
* Provides standard commands for devices.
* Ensures hardware resources are properly used (e.g., memory, interrupts).

**🔹 Types of Device Drivers**

1. **Kernel-mode drivers**
   * Run inside the OS kernel (low-level).
   * Have direct access to hardware.
   * Examples: File system drivers, USB drivers, Network card drivers.
2. **User-mode drivers**
   * Run in user space, safer than kernel drivers.
   * Cannot directly access hardware → must go through kernel.
   * Examples: Printer drivers, Camera drivers.
3. **Virtual device drivers (VxD)**
   * Used in older Windows versions.
   * Allow multiple apps to share hardware.
   * Example: Virtual printer drivers.
4. **Firmware drivers**
   * Stored in hardware itself (inside ROM/Flash).
   * Example: BIOS drivers, GPU firmware.
5. **Plug and Play drivers**
   * Auto-detected by Windows when new hardware is connected.
   * Example: USB flash drive.

**2. Packaging (Application Packaging)**

**🔹 What is Packaging?**

* **Application Packaging** is the process of preparing software in a **standardized, distributable format** so it can be easily installed, updated, or removed on many systems.
* It is widely used in enterprise IT to ensure consistency and reduce errors in software deployment.

**🔹 Why Packaging?**

* Manual installation is time-consuming and error-prone.
* Packaging makes deployment **faster, uniform, and easier to maintain**.

**🔹 Common Packaging Formats**

1. **MSI (Microsoft Installer)** → Standard Windows installer file.
2. **EXE (Executable)** → Traditional installer file, but less controlled.
3. **MSIX / AppX** → Modern packaging format for Windows 10/11, supports containerization.
4. **App-V** → Virtual application packaging by Microsoft.

**🔹 Steps in Application Packaging**

1. **Capture** – Record changes made by manual installation.
2. **Customize** – Apply organization policies (shortcuts, registry, silent install).
3. **Test** – Verify package works in different environments.
4. **Deploy** – Deliver package via SCCM, Intune, or other deployment tools.

**3. Self Repair (MSI Feature)**

**🔹 What it is**

* A feature of **Windows Installer (MSI)** that can **repair missing or corrupted components** of an application automatically.
* Triggered when a required resource (file, registry, COM entry) is missing.

**🔹 How it works**

* MSI has **KeyPath** for each component (important file/registry entry).
* If KeyPath is missing or broken → Self-Repair is triggered.
* Example: You delete a DLL → Next time app runs, MSI reinstalls the DLL.

**🔹 When it is triggered**

* When application is launched.
* When advertised shortcut is clicked.
* When COM object of the application is called.

**🔹 Benefits**

* Reduces corruption issues.
* Ensures application integrity.
* Minimizes helpdesk calls.

**4. Active Setup**

**🔹 What is Active Setup?**

* A Windows mechanism to **install or repair per-user settings** the first time a new user logs in.
* Useful when an app needs **registry keys or files in user profile**.

**🔹 How it works**

* Configurations are stored in registry:
  + HKLM\Software\Microsoft\Active Setup\Installed Components (machine-wide)
  + HKCU\Software\Microsoft\Active Setup\Installed Components (user-specific)
* When a new user logs in → Windows compares HKLM entries with HKCU.
* If entry exists in HKLM but not in HKCU → it runs the command defined in StubPath.

**🔹 Example**

* Application installs Active Setup entry in HKLM with a script.
* When a new user logs in, script runs automatically → sets registry keys or copies files.

**🔹 Uses**

* Apply per-user registry changes.
* Deploy user profile configurations (e.g., Outlook profile, application settings).

**5. Excel Add-ins**

**🔹 What they are**

* **Excel Add-ins** are small programs or extensions that add extra functionality to Microsoft Excel.
* They are usually written in VBA (Visual Basic for Applications), COM, or as Office add-ins (JavaScript).

**🔹 Types of Add-ins**

1. **Excel Add-in (.xlam)** → Custom functions/macros created by users.
2. **COM Add-in (.dll, .exe)** → Written in languages like C# or VB.NET.
3. **Office Store Add-ins** → Modern add-ins built using web technologies (HTML, JS).

**🔹 How to enable/disable**

* File → Options → Add-ins → Manage Excel Add-ins.
* Select checkboxes to enable or disable.

**🔹 Examples**

* **Analysis ToolPak** → Provides advanced data analysis functions.
* **Solver Add-in** → Helps in optimization and equation solving.
* **Power Query / Power Pivot** → For data modeling and BI tasks.

**🔹 Uses**

* Automate repetitive tasks.
* Extend Excel’s built-in functions.
* Connect Excel to external data sources.
* Useful in finance, data science, reporting.

**11/08/25**

**1. File and Registry Permissions**

**🔹 What it is**

* Windows protects its files and registry entries using NTFS permissions (for files/folders) and Registry ACLs (Access Control Lists).
* These permissions decide who can read, write, modify, or delete a file or registry entry.
* Critical for security, application stability, and preventing unauthorized access.

**🔹 File Permissions**

**Common NTFS permissions for files/folders:**

1. **Full Control →** Can read, write, modify, delete, and change permissions.
2. **Modify →** Can read, write, and delete but cannot change permissions.
3. **Read & Execute →** Can open and run files, but cannot edit.
4. **Read →** Only view contents**.**
5. **Write →** Add new content but cannot delete/modify existing.

**Example:**

* **An app config file →** user only gets *Read* permission.
* **Admin/installer →** has *Full Control* to update it.

**🔹 Registry Permissions**

**Similar concept, but for registry keys:**

* **Full Control** → Create, delete, modify keys/values.
* **Read →** Only view.
* **Special Permissions →** Fine-grained control like *Set Value*, *Query Value*.

**Example:**

* **HKEY\_LOCAL\_MACHINE (HKLM) →** usually requires Admin/System **rights.**
* **HKEY\_CURRENT\_USER (HKCU) →** controlled by the logged-in user.

**🔹 Importance**

* Prevents malware or users from corrupting critical system files.
* Ensures only trusted apps can modify registry entries.
* Used in Application Packaging to decide proper install/uninstall behavior.

**2. Lockdown**

**🔹 What it means**

* “Lockdown” refers to restricting permissions on files, folders, or registry keys after installation.
* Done so that end-users cannot modify critical resources required by an application.

**🔹 Why it is needed**

* Applications often store configs, executables, and libraries.
* If users modify/delete them, application may break.
* Lockdown ensures security + stability.

**🔹 How it is applied**

* **By setting NTFS permissions (DACL – Discretionary Access Control List).**
* **Example:**
  + **Application’s .exe file →** *Read & Execute* for users, *Full Control* only for Admin/System.
  + **Registry settings like license keys →** Read-only for users.

**🔹 Real-world example**

* **MS Office →** installed files are locked down, users can only run them**.**
* **Antivirus software** → prevents modification of its registry keys and service files.

**3. Difference between User, Admin, and System Context**

**🔹 Context = Security boundary**

When a process runs in Windows, it runs under a security context (User/Admin/System). This decides what permissions it has.

**🔹 User Context**

* **Runs under the logged-in user account.**
* **Permissions:**
  + Access to HKCU (user registry hive).
  + Read/Write to user’s profile folders (C:\Users\Username\...).
  + Limited access to system files (C:\Program Files, HKLM).
* **Examples:**
  + Opening Word, Chrome, Spotify.
  + Apps can only modify per-user settings.

**🔹 Admin Context**

* **Runs with elevated privileges.**
* **Permissions:**
  + Can access both HKCU and HKLM.
  + Can install/uninstall applications.
  + Can modify C:\Windows, C:\Program Files.
* **Examples:**
  + Installing software.
  + Running Command Prompt/PowerShell as *Administrator*.

**🔹 System Context**

* Also called LocalSystem account.
* Has higher privileges than Admin, used by the OS itself.
* **Permissions:**
  + Full access to all system files, services, registry.
  + Can perform tasks even Admin cannot.
* **Examples:**
  + Windows Services like *Winlogon, Task Scheduler, Windows Update*.
  + MSI installers running in SYSTEM context during enterprise deployment.

**🔹 Summary Table**

| **Context** | **Scope** | **Example Usage** | **Access** |  |
| --- | --- | --- | --- | --- |
| User | Current user only | Running apps like Chrome, Word | HKCU, User folders |  |
| Admin | Whole machine | Install/uninstall apps | HKLM, Program Files, Windows folders |  |
| System | Entire OS | Windows Services, deployments | Everything (highest) |  |

**12/08/2025**

**1. PowerShell Cmdlets**

**🔹 What are Cmdlets?**

* Cmdlets (Command-lets) are **lightweight commands** used in PowerShell.
* They are **.NET framework classes** designed to perform specific functions.
* Unlike traditional commands, cmdlets return **objects** (not plain text).
* These objects can be piped (|) into other cmdlets for further processing.

**🔹 Structure of a Cmdlet**

* Always in the form of **Verb-Noun**
  + Example: Get-Process, Set-ExecutionPolicy, New-Item
* Consistent naming makes it easy to guess functionality.

**🔹 Categories of Cmdlets**

1. **Get Cmdlets** – retrieve information
   * Get-Service, Get-Process, Get-Item
2. **Set Cmdlets** – modify/change configuration
   * Set-Date, Set-ExecutionPolicy
3. **New Cmdlets** – create new objects/resources
   * New-Item, New-LocalUser
4. **Remove Cmdlets** – delete objects
   * Remove-Item, Remove-Module
5. **Start/Stop Cmdlets** – control processes/services
   * Start-Service, Stop-Process

**🔹 Key Examples**

* **System Management**:
  + Get-Service → Lists services
  + Start-Service Spooler → Starts Print Spooler
* **Process Management**:
  + Get-Process → Shows running processes
  + Stop-Process -Id 1234 → Kills a process
* **File/Folder Management**:
  + Get-ChildItem C:\ → List files/folders
  + Remove-Item C:\file.txt → Delete file
* **Security**:
  + Get-ExecutionPolicy / Set-ExecutionPolicy
* **Networking**:
  + Test-Connection google.com (like ping)

**🔹 Pipeline Example**

Get-Process | Where-Object {$\_.CPU -gt 100} | Sort-Object CPU -Descending

👉 Lists processes using more than 100 CPU cycles, sorted by CPU usage.

**🔹 Advantages of Cmdlets**

* Consistent Verb-Noun naming
* Return objects → better automation
* Easy integration with scripts
* Thousands available out of the box

✅ **In short**: PowerShell cmdlets are the building blocks of automation in Windows. They help manage system, network, processes, and security in a very efficient way.

**2. Intune Application Packaging**

**🔹 What is Intune?**

* Microsoft Intune is a **cloud-based endpoint management solution**.
* It manages **apps, devices, and policies** across Windows, macOS, iOS, and Android.
* For applications, Intune requires packaging them in **specific formats** so they can be deployed to managed devices.

**🔹 Why Packaging is Needed in Intune**

* To **standardize deployment** across thousands of devices.
* To ensure apps **install silently** without user interaction.
* To enforce **security policies** (like only approved versions).
* To allow **updates and uninstalls** remotely.

**🔹 Application Package Types in Intune**

1. **MSI (Windows Installer)**
   * Most common enterprise format
   * Directly supported by Intune
   * Example: appname.msi
2. **EXE (Executable)**
   * Requires conversion to **IntuneWin format** using Microsoft Win32 Content Prep Tool
   * Example: appname.exe → appname.intunewin
3. **MSIX / APPX**
   * Modern Windows packaging format
   * Built-in support in Windows 10+
   * Easier updates & cleaner uninstalls
4. **Store Apps (Microsoft Store / UWP)**
   * Deployed directly from Microsoft Store for Business
5. **Mobile App Packages**
   * **iOS** → .ipa (through Apple Store)
   * **Android** → .apk (through Play Store or enterprise upload)

**🔹 Intune Win32 Packaging Process**

1. Download the **Win32 Content Prep Tool** from Microsoft.
2. Run the tool to convert .exe or .msi → .intunewin.
3. IntuneWinAppUtil.exe -c <source\_folder> -s setup.exe -o <output\_folder>
4. Upload the .intunewin file into Intune portal.
5. Define **install command** (e.g., setup.exe /quiet) and **uninstall command**.
6. Add **detection rules** (to confirm if app is installed).
7. Assign app to user/device groups for deployment.

**🔹 Key Features of Intune App Deployment**

* **Silent installation** (no user prompts)
* **Detection rules** → ensures app installs successfully
* **Dependencies** → install prerequisites before main app
* **Supersedence** → replace old app version with new
* **Reporting** → see installation success/fail logs

**🔹 Example**

Deploying Google Chrome Enterprise via Intune:

* Download .msi package from Google.
* Upload directly in Intune (no conversion needed).
* Set install command:
* msiexec /i "googlechromestandaloneenterprise.msi" /qn
* Set uninstall command:
* msiexec /x {ProductCode} /qn
* Configure detection: Registry key HKLM\Software\Google\Chrome.

**🔹 Benefits of Intune Packaging**

* Centralized management of apps
* Works across Windows, Android, iOS, macOS
* Ensures compliance (only approved versions run)
* Reduces manual IT effort

✅ **In short**: Intune Application Packaging converts apps into deployable formats (MSI, EXE → IntuneWin, MSIX, Store apps). It allows silent, automated, and secure app delivery across all devices in an organization.

**13/08/25**

**1. PowerShell App Deployment Toolkit (PSADT)**

**🔹 What it is**

* **PSADT** is a free, open-source framework built on **PowerShell**.
* It provides a set of **ready-to-use functions** that make software deployment easier, especially in **enterprise environments**.
* Commonly used in **SCCM, Intune, or manual deployments**.
* Helps standardize installations/uninstallations with **logging, user prompts, error handling, and custom logic**.

**🔹 Key Features**

1. **User Interaction**
   * Show welcome screens, warnings, or prompts before install.
   * Example: Ask user to close Outlook before installing an add-in.
2. **System Checks**
   * Detect running applications and force-close if needed.
   * Check OS version, architecture (x86/x64), or free disk space.
3. **Software Installation/Uninstallation**
   * Wrapper around MSI, EXE, or Script-based installs.
   * Example: Execute-MSI -Action Install -Path "app.msi"
4. **Logging & Reporting**
   * Generates detailed logs (easy for troubleshooting).
   * Consistent naming convention for enterprise use.
5. **Custom Actions**
   * Modify registry, copy files, set environment variables.
   * Flexible PowerShell scripting included.

**🔹 Advantages**

* Reduces scripting effort (no need to write from scratch).
* Enterprise-ready: integrates with **SCCM/Intune**.
* Standard framework → ensures all deployments follow same structure.
* Easy rollback or repair logic.

**🔹 Example Usage Flow**

1. User launches installer → toolkit shows welcome screen.
2. Checks system requirements.
3. Installs MSI silently in background.
4. Shows success/failure message.
5. Logs the entire process.

**🔹 Real-life Example**

Deploying **Adobe Reader** in 1000+ PCs:

* Detect old versions → uninstall.
* Install new version silently.
* Log everything.
* If installation fails → send error log to admin.

**2. MSI (Microsoft Installer / Windows Installer Package)**

**🔹 What it is**

* **MSI** is a **file format** (.msi) used for installing software on Windows.
* Managed by the **Windows Installer service**.
* Unlike a simple .exe, MSI is **structured** and follows rules.

**🔹 Internal Structure**

1. **Database Format**
   * MSI is basically a database of instructions.
   * Contains **tables** (like relational DB) → each table defines install actions.
2. **Key Tables**
   * **File Table** → which files to install.
   * **Registry Table** → registry keys to be created.
   * **Component Table** → groups of resources (files/registry).
   * **Feature Table** → logical feature sets (e.g., Word, Excel in MS Office).

**🔹 Key Features of MSI**

1. **Silent Installation**
   * msiexec /i app.msi /qn → no user interaction.
2. **Repair Functionality**
   * Can check if files/registry are missing and **self-repair**.
3. **Rollback**
   * If installation fails, MSI rolls back changes.
4. **Upgrades & Patches**
   * Supports incremental updates using **.msp** (patch files).
5. **Standardization**
   * Always behaves the same way → predictable for IT admins.

**🔹 Benefits**

* Reliable & structured installs.
* Easy to **deploy in enterprises** (via SCCM, Intune, Group Policy).
* Supports automation & scripting.
* Built-in repair and rollback → reduces broken installs.

**🔹 Example Commands**

* Install: msiexec /i app.msi
* Uninstall: msiexec /x app.msi
* Silent install: msiexec /i app.msi /qn
* Logging: msiexec /i app.msi /l\*v install.log

**🔹 Example Use Case**

Installing **Google Chrome MSI** across a company:

* Admins push via Intune.
* MSI installs silently in background.
* If a user deletes chrome.exe, Windows Installer automatically repairs it.

✅ **Summary**

* **PSADT** → A PowerShell toolkit that makes deployments smarter (user prompts, checks, logging).
* **MSI** → A structured installer package format with self-repair, rollback, and enterprise support.

**14/08/25**

**1. Process Explorer**

**🔹 What it is**

* A **free advanced task manager** from Sysinternals (Microsoft).
* Gives **real-time info** about processes, DLLs, handles, threads, CPU, GPU usage.
* More powerful than Windows Task Manager.

**🔹 Key Features**

* **Tree View** → See parent-child relation of processes.
* **DLL View** → Which DLLs a process is using.
* **Handle View** → Which files/registry keys a process has opened.
* **Verify Signatures** → Check if executables are trusted.
* **Search** → Find which process is locking a file.

**🔹 Use Cases**

* Troubleshooting high CPU/memory usage.
* Detecting malware processes.
* Finding file/registry locks.
* Debugging application hangs.

**2. Errors in Repackager**

**🔹 What is Repackager**

* A tool used in **application packaging** to convert EXE → MSI.
* Tracks changes made during an installation.

**🔹 Common Errors**

1. **File Copy Errors** – Some files fail to capture due to permissions.
2. **Registry Capture Errors** – System-protected registry entries not recorded.
3. **Driver/Service Install Issues** – Repackager not suited for low-level drivers.
4. **64-bit vs 32-bit Conflicts** – Wrong architecture capture.
5. **Duplicate Keys/Files** – Same entry recorded multiple times.

**🔹 Fixing Errors**

* Run Repackager as **Admin/System**.
* Clean VM snapshot before capturing.
* Exclude system folders (Windows, ProgramData) unless needed.
* Validate MSI using tools like **Orca / ICE Validation**.

**3. WHOIS**

**🔹 What it is**

* A command/tool to **query domain information** from WHOIS servers.
* Gives ownership details of a domain or IP address.

**🔹 What Info It Provides**

* Domain registrar (GoDaddy, Namecheap, etc.)
* Registered owner or organization (unless privacy protected).
* Registration & expiration date.
* Name servers.
* Contact details (sometimes masked).

**🔹 Usage**

* In CMD/PowerShell: whois domain.com (needs whois utility installed).
* Online WHOIS lookup websites.

**🔹 Use Cases**

* Security: Check domain legitimacy.
* Networking: Troubleshoot domain issues.
* Legal/Compliance: Identify malicious websites.

**4. Sysmon (System Monitor)**

**🔹 What it is**

* Part of Sysinternals suite.
* A **Windows system service + driver** that logs detailed security-related events into **Event Viewer**.

**🔹 Key Capabilities**

* Process creation with hashes.
* Network connections (source/destination).
* File creation time changes.
* Loading of drivers & DLLs.
* Registry changes.

**🔹 Usage**

* Installed via: sysmon.exe -i sysmonconfig.xml
* Configurable with XML to choose what events to log.

**🔹 Use Cases**

* Threat hunting & incident response.
* Detecting persistence mechanisms (startup registry keys, services).
* Investigating malware activity.

**5. Regmon (Registry Monitor)**

**🔹 What it is**

* Legacy Sysinternals tool (now replaced by **Process Monitor**).
* Used to **monitor registry activity in real time**.

**🔹 What It Shows**

* Which process accessed registry keys.
* Read/Write/Delete operations.
* Success/Failure status of access.

**🔹 Use Cases**

* Debugging software installation registry writes.
* Identifying malware modifying registry.
* Tracking application crashes due to missing keys.

**18/08/2025  
  
1. MSIX Technology Fundamentals with Architecture**

**🔹 What is MSIX?**

* MSIX is Microsoft’s **modern packaging format** (introduced in Windows 10, 2018).
* It combines the best of **MSI, AppX, and App-V**, designed for **secure, reliable, and flexible deployment**.

**🔹 Fundamentals**

* **Universal Packaging** – Works across devices: desktop, laptop, tablets.
* **Container-based** – Apps run inside a container, isolating them from the OS.
* **Declarative Model** – Instead of custom actions, configurations are described in manifest files.
* **Digital Signing** – All MSIX packages must be signed → improves trust & security.

**🔹 MSIX Architecture**

1. **Package Manifest (AppxManifest.xml)**
   * Defines app identity, capabilities, dependencies.
   * Similar to metadata in MSI but more structured.
2. **Containerization Layer**
   * Separates app files from system files.
   * Prevents unwanted registry/file modifications.
3. **App Data Management**
   * Writes go to *app data storage locations* (not system folders).
   * Allows **clean uninstall** (no leftover files).
4. **Delivery + Update**
   * Supports incremental updates → only changed blocks are downloaded.

**2. MSIX Benefits and Why MSIX?**

**🔹 Benefits**

1. **Security**
   * Digital signature required.
   * Runs inside container → reduces malware risk.
2. **Clean Install/Uninstall**
   * No registry pollution or leftover files.
3. **Efficient Updates**
   * Uses block-level updates → saves bandwidth.
4. **Backward Compatibility**
   * Can package old Win32/NET apps.
5. **Cross-Platform Deployment**
   * Works with Intune, SCCM, Microsoft Store.
6. **Simplified Management**
   * Declarative model removes complex scripting.

**🔹 Why MSIX?**

* MSI is old → leaves residues, harder updates.
* AppX is limited → only UWP apps.
* App-V is complex → hard adoption.  
  👉 MSIX unifies all into **one modern solution**.

**3. Current Limitations of MSIX**

* Doesn’t support **drivers, services, or kernel-level components**.
* Some apps requiring deep OS integration may not run well.
* Conversion tools may struggle with **complex legacy apps**.
* Requires **Windows 10 (1809+) or Windows 11** → older systems not supported.
* Limited offline deployment support.

**4. Microsoft MSIX Packaging Tool and Other Editors**

**🔹 MSIX Packaging Tool**

* Free Microsoft app (from Store).
* Converts **existing MSI, EXE, App-V** into MSIX.
* Provides **GUI-based repackaging**.
* Captures installation changes → generates MSIX package.
* Allows package editing (name, version, manifest).

**🔹 Other Editors**

1. **Advanced Installer**
   * Third-party tool, supports MSI + MSIX packaging.
   * Rich UI, enterprise features.
2. **RayPack Studio**
   * Enterprise-grade repackaging tool.
   * Supports large-scale deployments.
3. **InstallShield**
   * Widely used commercial packaging tool.
   * MSIX support in latest versions.

**5. Overview of Microsoft System Center Configuration Manager (SCCM)**

**🔹 What is SCCM?**

* SCCM (now **Microsoft Endpoint Configuration Manager**) is an **enterprise management system**.
* Helps manage **Windows devices, applications, OS deployments, patches, and security policies**.

**🔹 Key Features**

* **Application Deployment** – Install/update apps at scale.
* **OS Deployment (OSD)** – Automated Windows installation.
* **Patch Management** – Distribute Microsoft + third-party updates.
* **Inventory** – Hardware & software inventory collection.
* **Compliance & Security** – Apply configurations, monitor compliance.
* **Integration** – Works with Intune for hybrid management.

**6. Planning and Deploying a Stand-Alone Primary Site**

**🔹 Stand-Alone Primary Site**

* A **single SCCM site** managing devices directly.
* No hierarchy (no CAS needed) → simpler.
* Suitable for **medium organizations**.

**🔹 Planning**

1. **Assess Needs**
   * Number of clients (up to 100k supported).
   * Network structure.
   * Deployment methods.
2. **Prepare Server Environment**
   * Windows Server with SQL Server.
   * AD integration recommended.
3. **Install Prerequisites**
   * .NET, IIS, BITS, WSUS (for updates).
4. **Deploy SCCM**
   * Run installer → choose Stand-alone Primary Site.
   * Configure site code, site name.
5. **Client Installation**
   * Push SCCM clients to managed devices.

**7. Managing Computers and Devices in the Enterprise**

**🔹 Why manage devices?**

* Ensures **security, compliance, and standardization** across the enterprise.

**🔹 Methods**

1. **SCCM Client Agent**
   * Installed on each computer.
   * Handles communication with SCCM server.
2. **Collections**
   * Group of devices based on queries (e.g., "All Laptops").
   * Used for targeting apps, updates.
3. **Policies**
   * Define settings → delivered via SCCM.
4. **Monitoring & Reporting**
   * Track compliance, software usage, update status.

**8. Preparing the Management Infrastructure to Support Desktop Devices**

**🔹 Key Steps**

1. **Active Directory Integration**
   * Organize devices into **OUs**.
   * Simplifies policy deployment.
2. **Networking Setup**
   * Ensure reliable **DNS, DHCP**.
   * Configure firewall rules for SCCM/Intune communication.
3. **Certificates & Security**
   * For secure client-server communication.
4. **Client Deployment Strategy**
   * Decide push, GPO, manual install.
5. **Monitoring + Updates**
   * Ensure WSUS or Windows Update for Business integrated.

**19/08/2025**

**📘 MSIX & SCCM Advanced Notes**

**1. Digital Signing Requirement and Implementation in MSIX**

**🔹 Why Signing is Needed**

* MSIX packages **must be digitally signed** to ensure authenticity.
* Signing proves:
  1. The package came from a **trusted source**.
  2. The content was not modified (tamper-proof).
  3. Required by Windows to install MSIX (unlike MSI, which can install unsigned).

**🔹 Certificate Types for Signing**

* **Self-Signed Certificate** → For testing purposes only.
* **Enterprise Certificate Authority (CA)** → For internal apps inside an organization.
* **Publicly Trusted Code Signing Certificate** → For apps to be published on Microsoft Store or shared widely.

**🔹 Implementation Steps**

1. Generate a certificate (self-signed or CA-issued).
2. Use **SignTool.exe** (Microsoft SDK) → Command:
3. signtool sign /a /f MyCert.pfx /p password MyApp.msix
4. Deploy the certificate to **Trusted Root Certification Authorities** on client machines (if not public).
5. Verify signature →
6. signtool verify /pa MyApp.msix

**2. Most Used Troubleshooting Tools for MSIX Packages**

**🔹 Common Tools**

1. **Event Viewer**
   * MSIX logs errors under *Applications and Services Logs → Microsoft → Windows → AppXDeployment-Server*.
   * Useful for debugging installation and deployment issues.
2. **PowerShell Cmdlets**
   * Add-AppxPackage → Installs an MSIX manually.
   * Get-AppxPackage → Checks if the package is installed.
   * Remove-AppxPackage → Uninstalls MSIX.
3. **MSIX Packaging Tool Logs**
   * Shows detailed capture/packaging errors.
4. **Process Monitor (ProcMon)**
   * Tracks registry and file access issues when MSIX fails.
5. **AppInstaller Logs**
   * Available in Event Viewer, helps with MSIX app update failures.

**3. Preparing the Management Infrastructure to Support Desktop Devices – SCCM**

**🔹 Key Requirements**

* **Active Directory Integration** → SCCM depends on AD for device/user discovery.
* **SQL Server** → Stores SCCM configuration and inventory.
* **Windows Server Roles** → Like WSUS for software updates.
* **Networking** → Proper DNS and IP addressing, open required ports.

**🔹 Infrastructure Setup Steps**

1. **Install SQL Server** → Backend database.
2. **Install SCCM Site Server** → Defines management scope.
3. **Configure Boundaries and Boundary Groups** → Ensure client machines can locate SCCM servers.
4. **Distribute Content** → Use Distribution Points (DPs) to host applications, updates, and OS images.

**4. Managing the Configuration Manager Client**

**🔹 What is SCCM Client?**

* A lightweight agent installed on managed PCs.
* Communicates with SCCM server to **receive policies, applications, updates, and compliance checks**.

**🔹 Key Management Tasks**

1. **Client Installation**
   * Methods: Client Push, Group Policy, Manual Install, Software Update Point.
   * Verify installation via Control Panel → Configuration Manager Properties.
2. **Client Health Monitoring**
   * SCCM regularly checks if the client is active and reporting.
   * Client Health dashboards show devices not reporting.
3. **Client Settings**
   * Define how often inventory is sent, power management rules, update schedules.
4. **Troubleshooting Client Issues**
   * Logs stored at C:\Windows\CCM\Logs.
   * Common logs:
     + ClientIDManagerStartup.log → Client registration.
     + PolicyAgent.log → Policy download.
     + CAS.log → Content access.

**5. Managing Inventory for PCs and Applications**

**🔹 Hardware Inventory**

* Collects details about PC hardware: CPU, RAM, disk space, OS version, etc.
* Runs on a schedule (default = every 7 days).

**🔹 Software Inventory**

* Scans files on client PCs (executables, scripts).
* Helps track installed applications.

**🔹 Asset Intelligence**

* Enhances software inventory with details like license usage, software categories.
* Helps IT know if apps are over/under-licensed.

**🔹 Compliance Settings**

* Define baselines → Example: Ensure antivirus is enabled, firewall is on.
* SCCM checks devices against compliance rules and reports non-compliance.