ChatGPT

Expanded Enterprise Packaging & Deployment MCQs

Application Packaging (MSI, MSIX, Intune Win32)

1. **Scenario:** You have a legacy application distributed as an MSI. However, the MSI is just a bootstrapper that launches a separate EXE with a GUI. When deployed silently (e.g. with `/qn`), `msiexec` returns success but the application never installs. What is the most likely cause?
2. A. The MSI is missing a required transform file.
3. B. The MSI is an administrative image meant only to extract files.
4. C. The bootstrap EXE requires user interaction and doesn't run silently.
5. D. The Windows Installer service is disabled on the target.

Answer: C
Explanation: Some MSIs merely wrap an EXE that presents its own UI. Running the MSI with `/qn` may skip running that EXE entirely, so nothing actually installs. In such cases you must repack or call the inner EXE with its silent switches (or use a proper install chain) to install the app.

1. **Scenario:** You packaged an MSI into a Win32 `.intunewin` app for Intune. The install command ran without errors, but Intune still shows "not detected" after completion. The detection rule uses the MSI's product code. What is the most likely reason?
2. A. Intune does not support MSI product-code detection for Win32 apps.
3. B. The installer required a reboot, and detection ran before reboot.
4. C. The product code changed when wrapping into `.intunewin`.
5. D. The Win32 Content Prep Tool invalidated the MSI's metadata.

Answer: B
Explanation: If the installer needs a reboot to finalize (for example, to install .NET or drivers), Intune's detection may run before the reboot finishes and not see the installed product. This causes a false "not detected" error. Ensuring proper reboot logic or using a file/registry detection that accounts for the restart will fix it.

1. **Scenario:** A vendor supplies your software as an MSIX package. You attempt installation on Windows 10 version 1803 and get an "unsupported" error. Why might this occur?
2. A. MSIX is only supported on Windows 11 and later.
3. B. You must enable the App Installer/MSIX feature on older Windows 10 versions.
4. C. The MSIX was built for ARM architecture.
5. D. The Universal C Runtime is not present by default on 1803.

Answer: B
Explanation: MSIX is supported on Windows 10 (1709 and later) [1], but earlier builds may not have the

necessary App Installer/MSIX support. On 1803 you often need to install or enable the MSIX/App Installer feature. Without it, the OS reports MSIX as unsupported.

1. **Scenario:** You add a prerequisite EXE to an existing Intune Win32 app package. How should you update the deployment?
2. A. Edit the existing `.intunewin` with a ZIP tool and add the EXE.
3. B. Re-run the Microsoft Win32 Content Prep Tool including the new EXE so it's bundled.
4. C. Upload the EXE separately as its own Win32 app in Intune.
5. D. Place the EXE alongside the `.intunewin` on Intune's distribution point.

Answer: B
Explanation: The `.intunewin` file is a packaged archive created by the Win32 Content Prep Tool. To include new files (like an extra EXE), you must re-run the prep tool. It zips and encrypts all specified installers. You cannot just unzip or modify the existing `.intunewin` directly.

1. **Scenario:** After repackaging an installer, an application fails at startup, complaining about a missing "msvcp140.dll". Dependency Walker confirms that DLL is missing. How should you resolve this?
2. A. Copy `msvcp140.dll` from another system into `C:\Windows\System32`.
3. B. Install the appropriate Visual C++ Redistributable package that includes that DLL.
4. C. Manually edit the EXE's imports to remove the dependency.
5. D. Ignore it; it's a false positive from Dependency Walker.

Answer: B
Explanation: `msvcp140.dll` is part of the Microsoft Visual C++ 2015-2019 runtime. Installing the correct VC++ Redistributable will place the necessary DLLs on the system. Dependency Walker's report here is accurate — the app truly needs that DLL at runtime.

Device Drivers and SetupAPI Logs

1. **Scenario:** Where does Windows 10/11 log detailed driver installation information by default?
2. A. `C:\Windows\INF\SetupAPI.log`
3. B. `C:\Windows\INF\SetupAPI.dev.log`
4. C. `C:\Windows\System32\DriverStore\Log\drvinst.log`
5. D. `C:\Windows\System32\LogFiles\DriverSetup\setupapi.log`

Answer: B
Explanation: In Windows Vista and later, SetupAPI writes driver install logs to `%SystemRoot%\INF\SetupAPI.dev.log` [2]. This plain-text log shows the steps and any errors during device driver installation, useful for troubleshooting.

1. **Scenario:** A new device driver installation fails. Device Manager shows Code 28 ("drivers not installed"). The `SetupAPI.dev.log` shows "0x80070BC2". What does this error indicate?
2. A. No compatible driver found for the device's hardware ID.
3. B. The INF is unsigned and blocked by driver signature enforcement.
4. C. The system requires a reboot to complete a pending driver operation.
5. D. The INF's `Version` section is malformed.

Answer: A
Explanation: Error code **0x80070BC2** (class installer denied request) typically means no matching driver was found for the device's ID, leading to Code 28. You need to provide the correct driver INF/sys (or ensure hardware IDs match) to resolve it.

1. **Scenario:** A third-party printer driver is signed with a vendor certificate not in Microsoft's CA chain. On Windows 11 it fails to install due to "Unknown Publisher". What is the best workaround?
2. A. Disable driver signature enforcement via Group Policy.
3. B. Add the vendor's certificate to **Trusted Publishers** on the machine.
4. C. Install the driver in test mode (`bcdedit /set testsigning on`).
5. D. Convert the driver to an unsigned driver pack.

Answer: B
Explanation: For third-party signed drivers that aren't WHQL, you can import the vendor's signing certificate into the local machine's **Trusted Publishers** store. Then Windows will treat that publisher as trusted [3] and install the driver normally. (Disabling signing enforcement is riskier and generally not recommended.)

COM/Office Add-ins and LoadBehavior

1. **Scenario:** An Outlook add-in's registry shows `LoadBehavior = 3` under `HKCU\Software\Microsoft\Office\Outlook\Addins\YourAddin`. What does this setting mean?
2. A. The add-in is disabled on startup.
3. B. The add-in will load automatically at Outlook startup (Boot load).
4. C. The add-in is loaded only on demand when needed.
5. D. The add-in is in the disabled list due to a crash.

Answer: B
Explanation: In Office add-in registry settings, `LoadBehavior = 3` corresponds to **boot load** (load at startup) [4]. A value of 9 would indicate demand load, etc. So this setting means Outlook should load the add-in when it launches.

1. **Scenario:** You want to install a COM component so that all users on the machine can use it in Office. Where should the COM registration entries go?

    - A. Under `HKEY_CURRENT_USER\Software\Classes\CLSID`
    - B. Under `HKEY_LOCAL_MACHINE\Software\Classes\CLSID`
    - C. Under `HKEY_CURRENT_USER\Software\MyApp\CLSID`
    - D. Under `HKEY_USERS\.DEFAULT\Software\Classes\CLSID`

    Answer: B
    Explanation: Per-machine COM registration belongs in **HKLM\Software\Classes** (and on 64-bit Windows for a 32-bit COM add-in, under `Wow6432Node`). HKCU would register it just for the current user. To make it available for **all users**, use HKLM.

2. **Scenario:** An Outlook add-in keeps getting disabled by the resiliency feature (performance optimization). You see its `ProgID` under the Resiliency `DoNotDisableAddinList` with a DWORD value of `0x00000002`. How do you ensure it stays enabled at startup?

    - A. Change its `LoadBehavior` to 2 in the registry.
    - B. Set the `DoNotDisableAddinList` value to `0x00000001` (Boot load) for that ProgID [5].
    - C. Reinstall the add-in under an administrator account.
    - D. Disable all other add-ins to avoid conflicts.

    Answer: B
    Explanation: The `DoNotDisableAddinList` registry key under the Office Resiliency settings uses specific values (hex 1 for Boot load, 2 for Demand load, etc.) [5]. Setting your add-in's ProgID value to `0x00000001` marks it as a boot-load add-in in the allow list, preventing Outlook from disabling it at startup.

Active Setup & Context Scripts

1. **Scenario:** You configure Active Setup by adding a key under `HKLM\Software\Microsoft\Active Setup\Installed Components\{GUID}` with a `StubPath` and `Version`. Only new user profiles are initialized by it; existing users see no effect. Why?

    - A. Active Setup only runs once per user per version. Existing users already have the older version logged.
    - B. The `StubPath` must be an `.exe`, not a script.
    - C. Active Setup ignores entries without an `IsInstalled` flag.
    - D. Active Setup runs only on domain-joined machines.

    Answer: A
    Explanation: Active Setup writes the version value into each user's profile when it runs. If an existing user has the same or higher version, it won't run again. You must **increment the Version value** in HKLM for Active Setup to re-execute for current users [6]. New users see it because they have no prior version.

2. **Scenario:** You deploy a script as a computer startup (RunOnce) entry to write some keys under `HKCU`. When users log in later, those keys never appear in their profile. Why not?

    - A. LocalGroupPolicy scripts only run on shutdown.
    - B. The script ran under SYSTEM, so `HKCU` referred to the default system profile.
    - C. The RunOnce key must be under `HKCU` for it to work.
    - D. Windows blocks registry writes in startup scripts by design.

    Answer: B
    Explanation: RunOnce or startup scripts in HKLM execute as **LocalSystem**. SYSTEM has its own `HKCU` (which is the Default user), so any writes to `HKCU` go to the wrong hive. To affect each actual user's `HKCU`, you would need to run the script in the user's context (for example, a user logon script or Active Setup).

PowerShell Scripting & PSADT

1. **Scenario:** You deploy a Win32 app via Intune that uses a PSADT script calling `Show-InstallationWelcome` (a GUI prompt). Users never see this prompt. Why?

    - A. Intune automatically suppresses all pop-ups in deployed scripts.
    - B. The script is running as SYSTEM, so no interactive session is available for the GUI.
    - C. `Show-InstallationWelcome` only works on Windows 10 version 21H1 or later.
    - D. The `-UseShellExecute` parameter must be true for GUI prompts to appear.

    Answer: B
    Explanation: By default, Win32 apps (including PSADT scripts) deployed via Intune run under the SYSTEM account. SYSTEM has no user interface, so any PSADT GUI calls (like `Show-InstallationWelcome`) won't be visible. To show UI, you must run the script under a user context or use the PSADT service UI functionality.

2. **Scenario:** A PSADT script uses `Start-Process -FilePath msiexec.exe -ArgumentList '/i MyApp.msi /qn'` without additional parameters. The MSI never seems to install. What is likely wrong?

    - A. The `/qn` parameter should be `/quiet`.
    - B. `Start-Process` without `-Wait` launches the process asynchronously and the script ends immediately.
    - C. `Start-Process` cannot launch system processes from PSADT.
    - D. You must use `Invoke-MSI` instead of `Start-Process`.

    Answer: B
    Explanation: By default, `Start-Process` launches and immediately returns control, without waiting. If the script then finishes or exits, the MSI may never complete. One should include `-Wait` (or use the PSADT `Execute-MSI` function) to ensure the script waits for the installer to finish. (Note: PSADT also provides a wrapper to handle this.)

3. **Scenario:** In PSADT, which built-in function is designed to install an MSI with logging and rollback support?

    - A. `Install-ApplicationMsi`
    - B. `Execute-MSI`
    - C. `Start-InstalledApplication`
    - D. `Invoke-MsiExec`

    Answer: B
    Explanation: The PSAppDeployToolkit provides an `Execute-MSI` function (also `Execute-MSI -Install`, etc.) to handle MSI installs. It calls `msiexec.exe` with proper switches, captures logs, and returns correct exit codes. This is preferred over raw `Start-Process`.

Windows Logon Scripts & Scheduled Tasks

1. **Scenario:** A Group Policy logon script to map network drives sometimes fails for users. What setting can help ensure network resources are ready before logon scripts run?

    - A. Enable “Always wait for the network at computer startup and logon.”
    - B. Set the script type to “PowerShell” in GPO.
    - C. Disable Fast Startup in Power Options.
    - D. Use a scheduled task instead of a logon script.

    Answer: A
    Explanation: If logon scripts run before the network stack is fully available, drive mappings can fail. The GPO **“Always wait for the network at computer startup and logon”** ensures the machine delays logon until the network is initialized, improving script reliability.

2. **Scenario:** You need a task to run a cleanup script every night at 2:00 AM, regardless of user logon. Which Task Scheduler settings are needed?

    - A. Trigger: Daily at 2:00 AM; Check “Run whether user is logged on or not”.
    - B. Trigger: At startup; in Actions use `schtasks.exe /run`.
    - C. Trigger: Weekly; Disable “Run only when user is logged on”.
    - D. Trigger: On idle; Set idle timeout to 120 minutes.

    Answer: A
    Explanation: To run at a specific time regardless of user, use a scheduled trigger (Daily at 2:00 AM) and set the task to **“Run whether user is logged on or not”**. This ensures it executes even if no one is logged in.

3. **Scenario:** You create a scheduled task that should trigger on any user’s logon. It only runs for the user who created it. What change will fix this?

    - A. In the Trigger, select “At log on of Any user” instead of a specific user.
    - B. Check “Run with highest privileges”.
    - C. Enable “Hidden” in the General tab.
    - D. Use an Event-based trigger instead.

    Answer: A
    Explanation: By default a new task trigger might be configured for a specific user. To have it run on logon of any user, explicitly choose **“At log on of Any user”** in the task’s trigger. Then every user’s logon will fire the task.

Intune Packaging Logic (.intunewin, Detection)

1. **Scenario:** An Intune Win32 app’s detection rule is set to look for a specific **HKCU** registry key. Testing shows it never detects the app as installed. Why?

    - A. Intune cannot use HKCU detection for system-context deployments.

- B. The Intune Management Extension doesn't have access to HKCU.
- C. HKCU keys are per-user and Intune's detection runs as SYSTEM by default.
- D. Group Policy is blocking registry reads in user scripts.

Answer: C
Explanation: By default, Intune installs Win32 apps as SYSTEM. A HKCU-based detection will run in the system context and therefore not see per-user keys (or will see the wrong user hive). For system deployments, use HKLM or file-based detection. For user-targeted installs, you may need a different approach.

2. **Scenario:** You package a Win32 installer into an `.intunewin` and upload to Intune. The App Information page requires an install and uninstall command. You provide `msiexec /i MyApp.msi /qn` for install. Later you realize you also need to silently remove a previous version. What should you do?

   - A. Set the Uninstall command to `msiexec /x {GUID} /qn` using the MSI product code.
   - B. Call the uninstall from inside the install command.
   - C. Upload a second `.intunewin` for uninstall.
   - D. Intune will automatically generate an uninstall based on MSI.

   Answer: A
   Explanation: You should explicitly enter an uninstall command, typically `msiexec /x <ProductCode> /qn`. Intune does not auto-generate it. This ensures the previous version is removed silently. The install and uninstall commands are fields you fill in on the App creation wizard.

3. **Scenario:** After a Win32 app deployment, devices report exit code `0x87D10102` ("not detected"). The app actually installed successfully, but Intune still retries. What common oversight causes this?

   - A. The uninstall command was incorrect.
   - B. The detection rule does not match what the installer actually creates.
   - C. The content was not correctly wrapped into `.intunewin`.
   - D. Intune requires a reboot policy even if the app didn't need it.

   Answer: B
   Explanation: Error `0x87D10102` usually means the detection rule didn't match the post-install state. For example, maybe you checked for a file that wasn't created, or used the wrong registry path/product code. Ensuring the detection precisely matches something the installer sets will fix the issue.

Troubleshooting Tools (ProcMon, Dependency Walker, etc.)

1. **Scenario:** An application fails to launch due to a missing DLL. You use Dependency Walker (depends.exe) and it reports many missing `api-ms-win-*` DLLs. What does this imply?

   - A. The app is incompatible with your OS.
   - B. Dependency Walker is showing Windows API set names, which are normal and not actual files [7].

- C. You need to copy those `api-ms-win-*.dll` files to the app folder.
- D. The Visual C++ runtime is missing.

Answer: B
Explanation: Dependency Walker often flags missing `api-ms-win-*.dll` entries, but these are **API sets** that Windows redirects internally. They aren't actual files to distribute. This is a known Dependency Walker artifact [7]. To diagnose a real missing DLL, use a runtime monitoring tool instead.

2. **Scenario:** You want to monitor an installer's actions on the registry and file system in real time. Which tool do you use?

   - A. Event Viewer
   - B. Sysinternals Process Monitor (ProcMon)
   - C. Windows Performance Monitor
   - D. MS Config

   Answer: B
   Explanation: Process Monitor (ProcMon) can capture real-time registry, file, and process activity [8]. It's ideal for seeing exactly what files/keys an installer is accessing or failing to find.

3. **Scenario:** You suspect a dependency issue with a Windows executable. Which approach will definitively reveal any missing DLLs at runtime?

   - A. Run `sfc /scannow`.
   - B. Use Process Monitor to watch for "NAME NOT FOUND" file errors when the app starts.
   - C. Reinstall the OS.
   - D. Call `regsvr32` on all expected DLLs.

   Answer: B
   Explanation: Process Monitor can filter for the process in question and show any attempts to open DLLs that result in "NAME NOT FOUND". This reveals exactly which DLL(s) are missing at runtime, whereas static tools like Dependency Walker may mislead with API-set entries.

4. **Scenario:** Dependency Walker flags that `KERNEL32.dll` depends on missing `API-MS-WIN-CORE-*.dll`. Should this be fixed by adding those DLLs?

   - A. Yes, copy the missing API-MS-WIN DLLs to the system folder.
   - B. No, these API sets are part of Windows and are resolved internally. This is a false positive in Dependency Walker.
   - C. Yes, install the Windows 7 runtime library.
   - D. No, only recompile the application.

   Answer: B
   Explanation: Modern Windows uses **API sets** internally. The missing `api-ms-win-*.dll` entries in Dependency Walker are expected; the OS maps them at load time. You do **not** manually provide these DLLs. They are resolved by the system's API set schema, so it's a false alarm from DW [7].

5. **Scenario:** A process unexpectedly crashes during installation. You attach ProcMon and see “ACCESS DENIED” on a registry key. What might this indicate?

    - A. The installer is requesting a reboot via registry.
    - B. The process lacks permission to write to that key, likely due to running without elevation.
    - C. The key is write-only and cannot be read.
    - D. The registry is corrupted.

    Answer: B
    Explanation: “ACCESS DENIED” in ProcMon usually means the process (often running as a non-admin) tried to write to a protected key (e.g. under `HKLM`). It suggests the installer needed elevation or should have targeted a user-scoped (HKCU) key instead.

6. **Scenario:** You suspect an installer looks at a registry value in `HKLM`. Which tool can help you verify what values it reads?

    - A. Registry Editor (regedit.exe) in logging mode.
    - B. Process Monitor, by filtering for the process and registry operations.
    - C. Performance Monitor, with a registry counter.
    - D. WSUS logs.

    Answer: B
    Explanation: ProcMon can be filtered to the installer’s process and registry events. You can see exactly which `HKLM` (or any) values it queries or modifies. This makes it easy to confirm if it’s reading the correct path or being redirected.

7. **Scenario:** Dependency Walker reports that an executable is missing the “api-ms-win-crt-locale-l1-1-0.dll” dependency. What is the best first step?

    - A. Install (or repair) the Universal C Runtime / Visual C++ Redistributable.
    - B. Manually download that DLL from the internet and place it in the folder.
    - C. Change the program’s manifest to not use the CRT.
    - D. It’s a false positive; run the program and see if it actually crashes.

    Answer: D
    Explanation: The `api-ms-win-crt-locale-l1-1-0.dll` is an API set name and normally part of the Universal C Runtime. Dependency Walker will show it as missing, but Windows provides these API set DLLs internally. Instead of chasing it, try running the program – if it errors, then install the proper VC++ Redistributable. Otherwise, these can often be ignored.

8. **Scenario:** You capture an installation with ProcMon and filter to the installer’s process. You observe it repeatedly trying to open `C:\MissingFolder\file.txt`. Why is this useful?

    - A. It shows a failure point; the installer expected that file but didn’t find it.
    - B. It indicates the current directory; relative paths use the installer’s folder.
    - C. It’s not useful; missing file events are normal noise.
    - D. It means the installer is scanning the disk.

Answer: A
Explanation: A "NAME NOT FOUND" on `file.txt` suggests the installer attempted to access that file (maybe for version checking or copying) but couldn't. This pinpoints a missing component or wrong path. From there you know to include or supply that file.

Each question above is designed to challenge real-world deployment and packaging scenarios. The correct answer and explanation follow each question. The explanations cite relevant documentation or common knowledge to justify the solutions (e.g., using **ProcMon** for troubleshooting [8] or understanding **LoadBehavior** values for Office add-ins [4]). These new MCQs expand on the original topics without repeating earlier questions, covering all requested areas with practical context.

---

[1] MSIX features and supported platforms - MSIX | Microsoft Learn
https://learn.microsoft.com/en-us/windows/msix/supported-platforms

[2] SetupAPI Device Installation Log Entries - Windows drivers | Microsoft Learn
https://learn.microsoft.com/en-us/windows-hardware/drivers/install/setupapi-device-installation-log-entries

[3] Signature Categories and Driver Installation - Windows drivers | Microsoft Learn
https://learn.microsoft.com/en-us/windows-hardware/drivers/install/signature-categories-and-driver-installation

[4] [5] Support for keeping add-ins enabled | Microsoft Learn
https://learn.microsoft.com/en-us/office/vba/outlook/concepts/getting-started/support-for-keeping-add-ins-enabled

[6] Active Setup Explained
https://helgeklein.com/blog/active-setup-explained/

[7] On API-MS-WIN-XXXXX.DLL, and Other Dependency Walker Glitches | Ofek's Visual C++ stuff
https://ofekshilon.com/2016/03/27/on-api-ms-win-xxxxx-dll-and-other-dependency-walker-glitches/

[8] Process Monitor - Sysinternals | Microsoft Learn
https://learn.microsoft.com/en-us/sysinternals/downloads/procmon