Administration Guide | PUBLIC
2026-04-10

SAPSetup Guide

SAPSetup 9.0

© 2026 SAP SE or an SAP affiliate company. All rights reserved.

Content

1 Features at a Glance. 4

2 Latest Information. 5

3 Glossary. 6

4 Planning the Installation of SAP Front End Components. 8

4.1 Installation Scenarios. 8

4.2 Hardware and Software Requirements. 10

4.3 Use Case: Installation and Maintenance. 11

5 Setup and Administration of the Installation Server 12

5.1 Setting Up the Installation Server. 14

5.2 Adding SAP Front End Components to the Installation Server. 17

5.3 Updating Products on the Installation Server 18

5.4 Deleting Products from the Installation Server. 19

5.5 Installation Packages: Creation and Maintenance. 20

Creating an Installation Package. 20

Configuring Packages and Event Scripts. 21

Changing the Package Content. 26

Deleting an Installation Package. 27

Creating and Deploying a Single-File Installer for Packages. 28

Creating a Package Definition File. 29

5.6 Patching the Installation Server. 30

5.7 Configuring the Local Security Handling. 31

5.8 Installing and Configuring the Automatic Workstation Update Service. 32

5.9 Controlling a Remote Workstation. 34

Collecting Log Files Remotely. 35

Executing a Process Remotely. 36

Listing Remote Processes. 37

5.10 Removing the Installation Server. 38

5.11 Command Line Parameters for the Installation Server. 38

6 Installing and Administering SAP Front End Components and Packages. 40

6.1 Installing SAP Front End Components. 40

6.2 Installing Packages Configured by the Administrator. 41

6.3 Installing Using the Windows Logon Script 42

6.4 Updating SAP Front End Components. 43

6.5 Uninstalling SAP Front End Components. 45
6.6 Repairing Installed SAP Front End Components. 46
6.7 Command Line Parameters for the Workstation. 47

7 Troubleshooting. 51
7.1 Log and Error Files. 52
7.2 Return Codes. 53

8 Scripting. 55
8.1 PowerShell and NwEngine. 55
NwEngine.Shell. 55
NwEngine.Context. 83
NwEngine.Context.Log. 90
NwEngine.Variables. 92
NwSapSetupATLCommon.TextFileParser. 94
8.2 Visual Basic Script Application Program Interfaces. 102
NwEngine.Shell. 105
NwEngine.Context. 129
NwEngine.Context.Log. 135
NwEngine.Variables. 137
NwSapSetupATLCommon.TextFileParser. 139

1 Features at a Glance

SAPSetup provides a fully featured lifecycle management for SAP front end components on Windows™ and an easy and reliable handling by using installation servers.

Front end components are available from an installation server or a distribution medium such as a DVD.

With SAPSetup you can:

- install new SAP front end components, uninstall existing ones, and update the remaining components in one cycle.
- maintain nearly all Windows™-based SAP front end components on a single installation server.
- integrate additional SAP front end components into an existing installation server.
- deploy multiple SAP front end components on workstations using a single command.
- manage easily the installation server using wizard-driven user interfaces.
- deploy SAP components on workstations without requiring administrative privileges on all machines, if you have configured Local Security Handling (LSH).
- automatically update the workstations and reboot them if necessary with the *Automatic Workstation Update Service*, whenever the installation server is patched, or the packages installed are updated.
- customize and extend the installation of packages (easier handling of multiple SAP front end components) using implemented script events.
- remotely access and control workstations on which you have administration privileges.
  You can collect remote log files, execute processes remotely, and list remote processes.
- automatically configure the directory for the creation of the installation server. By default, it is made NULL-session accessible. It can be shared automatically on the network for everyone to read.
- if the program executable `NwSapSetup.exe` is started from a batch file, you can catch the return code in an environment variable `<%ErrorLevel%>`.

2 Latest Information

The SAP Notes contain the most recent information about SAPSetup 9.0.

Make sure that you read the latest version of the following SAP Notes before your start the installation:

SAP Note Number	Description
1587566	Detailed release notes and known issues of SapSetup version 9.0. Description how to retrieve the latest version of SAPSetup.
1583967	General release information note for SAPSetup 9.0
1162270	Installing the DS installation service (analysis notes). Additional information on how to create and maintain an installation server to distribute the SAP Frontend Software for Windows to client computers using Local Security Handling.
1177282	End of support for SAP releases using Microsoft products.

You can find the SAP Notes in SAP Support Portal at https://support.sap.com/notes.

To check the latest version of this guide, see the SAP Support Portal athttps://support.sap.com/sltoolset *Front End Installation* > *Guide for SAPSetup* or the SAP Help Portal at https://help.sap.com/sapsetup.

3 Glossary

See the table below for descriptions of terms and abbreviations used in this guide

The are listed below.

Term	Description
AWUS	Automatic workstation update service. For more information, see Installing and Configuring the Automatic Workstation Update Service [page 32].
distribution medium	Contains the installer for an SAP front end component. It is either a directory in the file system or a single-file installer.
event script	A Visual Basic script that extends a package with custom actions. Events are installation, update and uninstall of a package. Event scripts can be defined to run before or after an event. For more information, see Configuring Packages and Event Scripts [page 21].
installation server	Central installation repository for distributing SAP front end components to many workstations. Prerequisite for packages, AWUS, and LSH.
LSH	Local security handling. For more information, see Configuring the Local Security Handling [page 31]
package	Selection of SAP front end components defined by the administrator of an installation server. Optionally contains configuration of installation parameters and event scripts.
patch	Single-file installer which contains only the parts of an SAP front end component that changed between two versions. A patch can be applied to a workstation or an installation server where the base version of the SAP front end component is already present.
product	Synonym for SAP front end component.
SAP front end component	SAP software that runs on a workstation and is installed and maintained with SAPSetup
SAPSetup	Tool suite for installing, updating, maintaining, and distributing software on Windows™.
single-file installer	Self-extracting executable that contains a package, a patch, or an installer for an SAP front end component.

Term	Description
update	Process of installing a newer version of an SAP front end component.
upgrade	Synonym for update.
workstation	Computer on which an SAP front end component is installed. The computer on which the installation server is running, is not considered as a workstation.

4 Planning the Installation of SAP Front End Components

This section describes how to plan the installation of SAP front end components.

1. You plan the installation scenario [page 8].
2. You make sure that your system meets the hardware and software requirements [page 10].
3. You review the installation and maintenance flow [page 11].

4.1 Installation Scenarios

You can install SAP front end components for Windows™ either from a distribution medium or from an installation server.

Installation Scenario	Description
Installation from a distribution medium (for example a DVD)	The administrator takes the distribution medium from workstation to workstation. This is mainly for test purpose or for standalone workstations.
Installation from an installation server	The administrator sets up an installation server, where the installation of the SAP front end components on different clients is started remotely. All necessary files are copied from the server to the clients during installation.

→ Tip

We recommend you to use the scenario *Installation from an installation server* because of its greater flexibility and efficiency, especially if many workstations are involved.

Installation from a Distribution Medium

A local installation on workstations with a distribution medium such as a DVD is useful for installing SAP front end components on single machines (for example, laptops) that are not connected to a Local Area Network (LAN). You can also use it for test purposes.

⚠ Restriction

This installation scenario has the following restrictions:

- There are no packages available.
- If patches need to be applied, each workstation has to be patched separately.
- An automatic workstation update is not supported.
- Local security handling (LSH) is not supported.

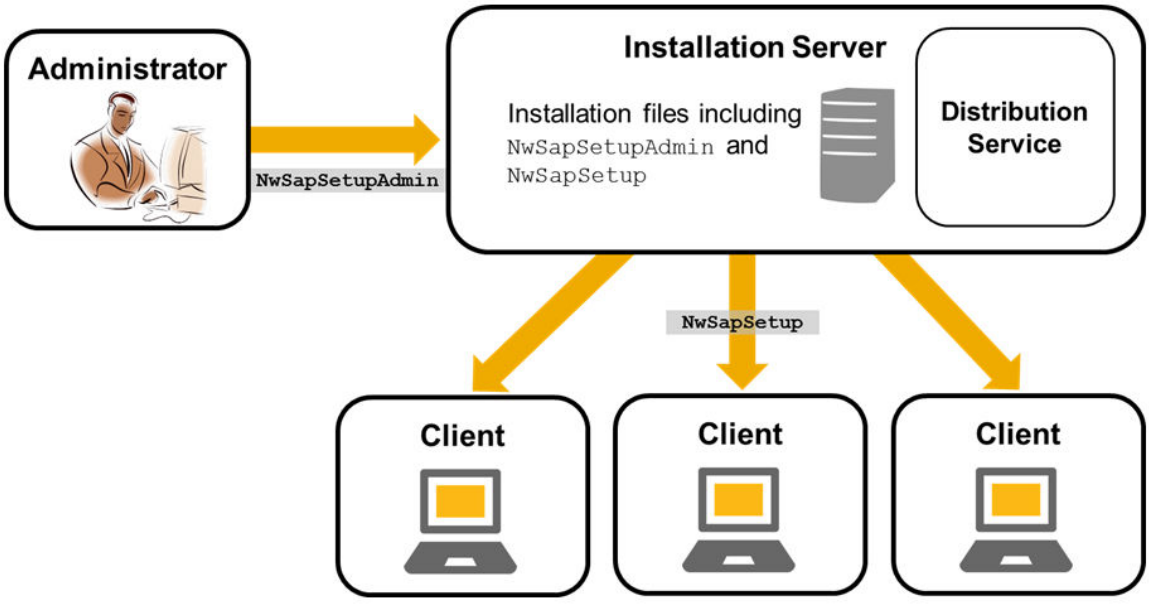
Installation from an Installation Server

The installation process from an installation server is flexible, easy, and customizable. It makes maintenance easier in any phase of the distribution process, for example, when patches are to be applied.

When installing SAP front end components with server-based workstation installation, the following options are available:

- Without user interaction (unattended)
- With user interaction (attended), where the user has the following options:
  - Select from installation packages that the administrator has configured
  - Select from a complete list of SAP front end components available on the installation server

The following figure shows how server-based installation works:



With an installation server, you as administrator can group various SAP front end components together as installation packages relevant for certain employee roles. You can also specify which package particular users should receive, or offer a variety of packages and allow the user to choose the most appropriate one.

In addition, you can configure the distribution service to add LSH functions to the installation server. With LSH, the installation can be started even by a user who is not a member of the local group of administrators. The distribution service then installs a service process on the client and starts `NwSapSetup.exe` in the context of this service. The user privileges are not altered. The installed service does not start other processes except `NwSapSetup.exe` from configured installation servers. Afterwards, the service removes itself from the workstation again.

The hard disk requirement on the installation server depends on the type and number of SAP front end components that are added to an installation server. For example, SAP GUI for Windows requires approximately 800 MB with all SAP front end components installed.

As administrator, you can configure your own installation packages on the installation server with `NwSapSetupAdmin.exe` .

To keep workstations up to date automatically, use the automatic workstation update service (AWUS). Whenever the installation server is patched, or packages are updated on the installation server, this service will update the workstation(s) and reboot them, if necessary. The AWUS works regardless of whether a user is logged on or not:

- If a user is logged on, the user is notified about the available update, and the update starts upon confirmation by the user. The user is also notified about whether a reboot is required, and the reboot is executed only upon confirmation by the user.
- If no user is logged on, the update and the reboot (if required) are started automatically.

Alternatively, you can apply patches on the installation server and then start the installation on your workstation to apply the patch on the workstation. You can control the installation using the command line. You can perform unattended installations with automatic patch installation on the workstation. To do so, insert the appropriate command line in the logon script of the user. The logon script is a program that is executed when a user logs on.

Related Information

Installing and Configuring the Automatic Workstation Update Service [page 32]

4.2 Hardware and Software Requirements

This section describes the requirements your system needs to meet.

Front end workstations

For hard drive clones only:

If the operating systems of your workstations were generated by hard drive cloning, make sure that the domain is set correctly. To do so, take the workstations out of the domain and then put them back in.

This is especially important if you intend to use LSH. More information: Configuring the Local Security Handling [page 31].

Computer for hosting the installation server

The following conditions have to be fulfilled:

- Windows server as operating system.
- .NET Framework 4.0 client profile has to be installed to run the administration tool.
- Read access for all users at all times, even after the installation is complete.
  This is required for maintenance purposes such as the distribution of patches and updates.
- Broadband network connection for high throughput.
- Enough free disk space for all SAP front end components to be imported.

4.3 Use Case: Installation and Maintenance

This section describes a typical installation and maintenance use case for installing SAP front end components on a workstation using an installation server.

1. You set up an installation server [page 14] and, if required, LSH.
   The installation server contains the following:
   - Installation programs
   - Configuration information (for example, packages)
   - Service files (for LSH)
   - Front end components to be installed
2. You maintain installation packages [page 20] for different user groups.
   Users can install multiple packages, and packages can share SAP front end components. You can configure installation parameters, such as installation directories, for the components that a package contains.
3. Recommendation: You configure LSH [page 31].
   Test the LSH by logging on to a user workstation as a user without local administrator privileges, and then running `NwSapSetup.exe` from the installation server.

   > **Note**
   >
   > Windows has local security mechanisms. Only users with local administrator privileges have write access to parts of the system database and the file system. SAPSetup solves this problem with the distribution service (DS) that resides on the server, and with the installation service (IS) that is installed on the workstation temporarily. The IS starts a new instance of `NwSapSetup.exe` that runs with sufficient privileges.

4. You install packages using the logon script [page 42] of your workstation or using other distribution mechanisms.
5. You patch the installation server [page 30].
6. You update SAP front end components [page 40] on a workstation when a new release is available on the installation server.

5 Setup and Administration of the Installation Server

This section describes how to set up and maintain an installation server for the efficient distribution of SAP front end components across workstations.

Features

The tool for administering the installation server is `NwSapSetupAdmin.exe`. You can find `NwSapSetupAdmin.exe` in the installation server's `Setup` directory after successful installation.

> **Note**
>
> `NwSapSetupAdmin.exe` requires .NET Framework 4, which you can download from the Microsoft Download Center.

- Product Import
  Use this feature to add new SAP front end components to an existing installation server for distribution over the network.
  More information: Adding SAP Front End Components to the Installation Server [page 17]
- Product Export
  Use this feature to export SAP front end components available on one installation server to another.
  More information: Updating Products on the Installation Server [page 18]
- Product Deletion
  Use this feature to delete a SAP front end component from an installation server.
  More information: Deleting Products from the Installation Server [page 19]
- Package Creation
  Use this feature to create packages for deployment. Packages can contain multiple SAP front end components, and their installation parameters can be customized.
  More information: Creating an Installation Package [page 20]
- Single-File Installer Creation
  Use this feature to create single-file installers.
  More information: Creating and Deploying a Single-File Installer for Packages [page 28]
- Package Definition Files (PDF) Creation
  Use this feature to create a package definition file (PDF) for an installation server package. PDF is a package description format that simplifies the interoperability and information exchange between SAPSetup and system management products such as Microsoft's System Management Server.
  More information: Creating a Package Definition File [page 29]
- Package Configuration
  Use this feature to change the attributes or the content of your packages.
  More information: Changing the Package Content [page 26]
- Installation Server Patch
  Use this feature to patch SAP front end components that are available on the installation server.

More information: Patching the Installation Server [page 30]

- Local Security Handling (LSH) Configuration
  Using this feature, workstation users are allowed to install SAP front end components from the installation server without requiring administrator privileges.
  More information: Configuring the Local Security Handling [page 31]
- Automatic Workstation Update Configuration
  Whenever the installation server is patched or the installed packages are updated, this service updates the workstations and reboots them, if required. AWUS works regardless of whether a user is logged on.
  - If a user is logged on, he is informed of the availability of updates and the update is started upon confirmation by the user. The user is also informed about whether a reboot is required (the reboot needs to be confirmed as well by the user).
  - If no user is logged on, the system performs the update and the reboot (if required) automatically.

  More information: Installing and Configuring the Automatic Workstation Update Service [page 32]
- Remote Workstation Control
  This feature uses the Windows Management Instrumentation (WMI) to help you remotely access and control workstations on which you have administrator privileges. You access this feature using the *Remote* menu, where the following options are offered:
  - Collect Log Files
  - Execute Process Remotely
  - Remote Task Manager

  More information: Controlling a Remote Workstation [page 34]

Tools

> **Note**
>
> The log files of all installation tools are stored in the following directory:
>
> `%ProgramFiles%\SAP\SapSetup\LOGs` (32bit Windows)
>
> `%ProgramFiles(x86)%\SAP\SapSetup\LOGs` (64bit Windows)
>
> Each tool stores the last 20 log files.

Tool	Description
`NwCheckWorkstation.exe`	Verifies the integrity of installed SAP front end components on a workstation Logfile: `NwCheckWorkstation.log`
`NwCreateInstServer.exe`	Creates a new installation server Logfile: `NwCreateInstServer.log`
`NwSapSetup.exe`	Executes an installation, patch, update or uninstall Logfile: `NwSapSetup.log`

Tool	Description
`NwSapSetupAdmin.exe`	Manages the installation server Logfile: `NwSapSetupAdmin.log`
`NwSapSetupIs.exe`	Runs as a Windows service on the workstation temporarily when using LSH (Local Security Handling), never to be executed manually Logfile: `NwSapSetupIs.log`
`NwSapSetupOnRebootInstSvc.exe`	Carries out the necessary steps to complete the installation (or patch, update, uninstall) as a Windows service if a reboot is needed, never to be executed manually Logfile: `NwSapSetupOnRebootInstSvc.log`
`NwUpdateInstServer.exe`	• Imports products into a freshly created installation server (called directly by `NwCreateInstServer.exe`) initially • Adds an additional product to an existing installation server • Updates products on an installation server with a newer version Logfile: `NwUpdateInstServer.log`

Procedure

1. You set up an installation server [page 14].
2. You add new SAP front end components to an existing installation [page 17], if required.
3. You update products on the installation server [page 18] with a newer version using a patch, if required.
4. You create and maintain installation packages [page 20], if required.
5. You patch the installation server [page 30], if required.
6. You configure LSH [page 31], if required.

5.1 Setting Up the Installation Server

This procedure helps you to distribute SAP front end software on multiple workstations across the network.

Prerequisites

To carry out the following task, you need to have local administrator privileges.

Context

`NwCreateInstServer` is a wizard-driven tool that helps the administrator to create a new installation server. All SAP front end components from the source are imported into the newly created installation server. The installation server consists of a directory that is shared on the network. There is no background process running.

Procedure

1. If you have an installation medium, start `NwCreateInstServer.exe` from the `Setup` directory. If you have a single file installer, open a command prompt and call it with the parameter `/CreateServer`. On the welcome screen, choose *Next*.

   You can choose *Cancel* at any stage to abort the process.
2. Choose the target directory, in which the installation server will be created.
   a. Choose *Browse* to navigate to the directory in which you want to create the installation server. Ensure that the directory is empty.
   b. Choose *Verify* to ensure that the chosen directory meets the following prerequisites:
      - It must exist.
      - It must be accessible to the administrator with full access.
   c. If you want to use one of the following features, choose *Share* to provide read access for the selected directory to the users that shall install from this installation server:
      - AWUS [page 32]
      - LSH [page 31]
      - Distribution using logon scripts.
      - Installation and update on workstations using the network share.
      - Execute installations from the installation server on remote workstations (remote execution [page 36]).

        > **Note**
        >
        > If you run `NwCreateInstServer.exe` with parameter `/silent`, the directory is automatically configured as a NULL-session share and is accessible on the network for everyone to read. If you do not require this automatic configuration, add the parameter `/DontConfigureServerPath` to the command line. With the parameter `/silent` you also need to specify the parameter `/dest:<destination directory>`, where `<destination directory>` is the directory in which the installation server will be created.
   d. Choose *Next* to continue.

> **Note**
>
> Processing is recorded in the file `NwCreateInstServer.log` located in the SAPSetup log file directory.
>
> If an error occurs, see Troubleshooting [page 51], section *Troubleshooting on the Installation Server*.

The wizard displays the progress of the server creation process and confirms the successful installation of the installation server.

3. To update the installation server with SAP front end components, choose *Next*.

   `NwCreateInstServer.exe` automatically calls `NwUpdateInstServer.exe` to import the SAP front end components into your installation server.

   The wizard displays the progress of the installation server update.

   > ⓘ Note
   >
   > Processing is recorded in the file `NwUpdateInstServer.log` located in the SAPSetup log file directory. If an error occurs, see Troubleshooting [page 51], section *Troubleshooting on the Installation Server*.

   `NwUpdateInstServer.exe` imports all SAP front end components available on the distribution medium into the installation server.

   After completing the update, the wizard confirms that you can use the installation server for installing SAP front end components on workstations.

4. If you have installed `.NET Framework` version 4, you can choose that the administration tool for the installation server will be started. To finish the update, choose *Close*.

Results

If you chose to start the administration tool for the installation server, `NwSapSetupAdmin.exe` is started automatically from the installation server. You can start it manually from the `Setup` directory of the installation server.

For more information, see Setup and Administration of the Installation Server [page 12].

> ⓘ Note
>
> - If you require a copy of the installation server, you can replicate the server by entering the following command:
>
>   ```
>   \\InstallationServerShare\Setup\NwCreateInstServer.exe /
>   Dest=<ReplicationPath> /Nodlg
>   ```
>
> - If you want to copy the SAP front end components and packages of one installation server to another one, enter the following command:
>
>   ```
>   \\InstallationServerShare\Setup\NwUpdateInstServer.exe /
>   Dest=<DestinationServer>\Setup /Nodlg
>   ```
>
>   The settings for AWUS, LSH and share permissions are not copied. Existing packages on the destination server will only be overwritten, if they have the same GUID. Avoid having two packages that have the same command line name.

→ Recommendation

After setting up your installation server, we recommend that you look for the latest patches. Obtain the latest SAPSetup version from SAP Note 1587566. For more information, see Patching the Installation Server [page 30].

5.2 Adding SAP Front End Components to the Installation Server

This procedure describes how to transfer new SAP front end components from a distribution medium to your installation server. From there, you can deploy the components to workstations on the network.

Prerequisites

- Check the disk space upfront.
- Ensure that the installation server is not in use during processing. If you are in doubt, you can remove the network share before starting this process, and create it again afterwards.
- If the components were downloaded as a single-file installer it must be extracted beforehand. Execute it on the command line with parameter `/extract:dest_dir` in order to extract it to directory `dest_dir`. If the directory does not exist, it will be created. Choose the directory `dest_dir` in the Product Import wizard.

Context

You can add SAP front end components using either `NwUpdateInstServer.exe` or `NwSapSetupAdmin.exe` (using the Product Import wizard). You can find these executables in the `Setup` directory of the source medium.

Procedure

To add SAP front end components to the installation server using:

- `NwUpdateInstServer.exe`
  1. Start `NwUpdateInstServer.exe` from the `Setup` directory of the distribution medium or the installation server that contains the SAP front end component you want to add.
  2. Supply the path to the installation server that you want to update.
  3. Follow the wizard instructions to update the server with SAP front end components that are available on the distribution medium.

- `NwSAPSetupAdmin.exe`
  1. Start `NwSAPSetupAdmin.exe` from the `Setup` directory of the installation server that you want to update with the new SAP front end components.
  2. In the toolbar, choose *Import Products*.
  3. Follow the wizard instructions and supply the path to the distribution medium or installation server containing the SAP front end component you want to add.

Example

You can add the Adobe LiveCycle Designer to the installation server. The Adobe LiveCycle Designer is delivered on the same DVD as SAP GUI for Windows. Start the program `<DVD Drive>:\ADOBE_LC_<release>\Setup\NwUpdateInstServer.exe`. After the welcome page, you are asked for the path to the installation server that you want to update. Navigate to the path of your installation server.

Choose *Next* and wait until the process is completed. Afterwards, the installation server administration tool `NwSapSetupAdmin.exe` starts. The added SAP front end component Adobe LiveCycle Designer is displayed on the *Products* tab.

You can now install Adobe LiveCycle Designer by starting `<installation_server_share>\SetupAll.exe` and selecting the SAP front end component to be installed, in this case the Adobe LiveCycle Designer.

5.3 Updating Products on the Installation Server

This sections describes how to update your installation server if you require a new release of SAP front end components.

Prerequisites

- Check the disk space upfront.
- Ensure that the installation server is not in use during processing the update. If you are in doubt, disable the network share before starting this process, and enable it again afterwards.
- A single-file installer must be extracted beforehand. Execute it on the command line with parameter `/extract:dest_dir` in order to extract it to directory `dest_dir`. If the directory does not exist, it will be created. Choose the directory `dest_dir` in the Product Import Wizard.

Context

Start `NwUpdateInstServer.exe` from the `Setup` directory of your distribution medium and follow the wizard instructions.

You can update SAP front end components using either `NwUpdateInstServer.exe` (the Product Import Wizard) or the command window.

Procedure

To update SAP front end components on the installation server using:

- `NwUpdateInstServer.exe`
  1. Start `NwUpdateInstServer.exe` from the `Setup` directory of the distribution medium.
  2. Follow the wizard instructions.
- **Command window**
  - Enter

    ```
    \\UpdateSource\Setup\NwUpdateInstServer.exe /dest=<installation server
    Setup directory> /nodlg
    ```

    or

    ```
    \\UpdateSource\Setup\NwUpdateInstServer.exe /dest=<installation server
    Setup directory> /silent
    ```

Related Information

Command Line Parameters for the Installation Server [page 38]

5.4 Deleting Products from the Installation Server

This section describes how to delete SAP front end components from an installation server.

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of the installation server from which you want to delete a SAP front end component.
2. On the *Products* tab, select the SAP front end component to be deleted and choose *Delete Product* from the context menu.
   The SAP front end component deletion wizard is displayed.
3. To delete a SAP front end component, follow the deletion wizard instructions.

   At the end of the deletion process, a message is displayed that informs you whether the deletion process has completed successfully.

5.5 Installation Packages: Creation and Maintenance

After setting up an installation server, you can create packages using `NwSapSetupAdmin.exe` in the `Setup` directory on the installation server.

A package consists of one or more SAP front end components that you want to install on the workstations. You can preconfigure the installation parameters of the package, for example the installation target directory. You can customize and extend the installation of packages by implementing script events.

5.5.1 Creating an Installation Package

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of your installation server.
2. To start the Package Creation Wizard, choose *New Package* from the toolbar.
3. To continue, choose *Next*.

   The wizard displays the SAP front end components available on the installation server.
4. Select the SAP front end components that you want to include in your package and choose *Next*.

   A yellow dot indicates a change in the selection list. A green plus sign indicates the SAP front end components to be installed.

   > ⓘ Note
   >
   > You can create packages that do not contain any SAP front end components. Such packages can be used to execute event scripts on workstations, for example to distribute your own configuration files.
   >
   > For more information, see Configuring Packages and Event Scripts [page 21] and Creating and Deploying a Single-File Installer for Packages [page 28].
5. Enter a display name for the new package and choose *Next*.
6. Enter a command line name for the new package and choose *Next*.

   This name is required when installing the package by calling `NwSapSetup.exe` on the command line with the parameter `/package`. For more information, see Command Line Parameters for the Workstation [page 47]

   The new package is created and a confirmation is displayed in the wizard.
7. Choose *Close*.

Results

The new package is displayed on the *Packages* tab. You can now configure the package to implement event scripts and customize installation parameters, for example the installation target path.

Related Information

Configuring Packages and Event Scripts [page 21]

5.5.2 Configuring Packages and Event Scripts

This section describes how to define a description, how to change the attributes of a package (for example, the name) and how to customize the installation of the package by adding scripts that are executed during events in its lifecycle.

Prerequisites

- An installation server is available.
- Packages are available.

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of the installation server.
2. Choose the *Packages* tab page.
   The existing packages are displayed on the left side of the screen.
3. Select the package that you want to customize.

   You can change its name, add a description, or implement script events.

   > ⓘ Note
   >
   > The end user will see name and description when he installs the package. Therefore, specify a name and a description that enables the end user to identify the correct packages.

4. **Optional:** To perform custom actions on the user's workstation during the installation of the package (for example, copying additional files), add event scripts. Select the *Configure Packages* tab and choose *Insert Script*.

   You can insert package event script samples delivered by SAP with the *Insert Script* function and adapt them according to your needs. The scripts are executed at the following events:

   - `On Installation Start`: executed before the installation of a package

- `On Installation End`: executed after the installation of a package
- `On Uninstallation Start`: executed before the uninstallation of a package
- `On Uninstallation End`: executed after the uninstallation of a package
- `On Update Start`: executed before the update and the repair of a package
- `On Update End`: executed after the update and the repair of a package

> ⓘ Note
>
> When uninstalling with the command line options `/uninstall /all`, the event scripts defined for the events `On Uninstallation Start` and `On Uninstallation End` will also be executed for all packages that are being uninstalled.

5. Choose *Save*.

Related Information

Package Event Script Samples [page 22]
Repairing Installed SAP Front End Components [page 46]
Changing the Package Content [page 26]

5.5.2.1 Package Event Script Samples

The following Visual Basic Script commands help you to customize your packages.

> → Tip
>
> All script samples below are available directly in `NwSapSetupAdmin.exe` on the *Configure Packages* tab.

Add custom logging to the log file

```
NwEngine.Context.Log.Write "Event: On Installation Start"
```

Log an error

```
‘Adds an error log entry to NwSapSetup.log, SAPSetup will exit with
'return code different than 0
NwEngine.Context.Log.WriteError "Error in custom script: something failed!"
```

Get the value of an installation variable

```
srcFile = NwEngine.Variables.ResolveString(_
"%SapSrcDir%\CustomerFiles\TestFile.txt")srcDir =
NwEngine.Variables.ResolveString("%SapSrcDir%\CustomerFiles")destDir =
"C:\temp\TestDir" destFile = "C:\temp\TestDir\TestFile.txt" destFile1 =
NwEngine.Variables.ResolveString("%ProgramFiles%\TestFile.txt") destFile2 =
NwEngine.Variables.ResolveString("%CommonProgramFiles%\TestFile.txt") destFile3
= NwEngine.Variables.ResolveString("%WinDir%\TestFile.txt")
```

Check whether a file exists

```
If NwEngine.Shell.FileExist( strSrcFile ) Then    'Do something
End If
```

Copy a file

```
'Copy a file. If the target file exists already, it will not be overwritten
NwEngine.Shell.CopyFile(srcFile, destFile)
'enforce overwriting an existing file
NwEngine.Shell.CopyFileEx(srcFile, destFile, vbTrue)
```

Delete a file

```
NwEngine.Shell.DeleteFile(destFile)
```

Copy a directory recursively

```
NwEngine.Shell.CopyDirectory(srcDir, destDir)
'enforce overwriting existing directories
NwEngine.Shell.CopyDirectoryEx(srcDir, destDir, vbTrue)
```

Create a directory

```
NwEngine.Shell.CreateDirectory(destDir)
```

Delete a directory

```
NwEngine.Shell.DeleteDirectory(destDir)
```

Check whether a registry key exists

```
If NwEngine.Shell.RegKeyExist("HKLM\SOFTWARE\SAP\TestKey") Then
    'Do something
End If
```

Set or create a registry value

```
NwEngine.Shell.SetRegValue("HKLM\SOFTWARE\SAP\TestKey\TestString", _
"REG_SZ", "TestValue")
NwEngine.Shell.SetRegValue("HKLM\SOFTWARE\SAP\TestKey\TestDWord", _
"REG_DWORD", "65536")
```

Read a value from the registry

```
regValue = _
NWEngine.Shell.GetRegValue("HKLM\SOFTWARE\SAP\TestKey\TestString")
```

Execute another application

```
‘child processes will have administrative rights on the current computercmdLine
= NwEngine.Variables.ResolveString("%WinDir%\NotePad.exe")
noWaiting = vbFalse
'This means: Wait for the started process
NwEngine.Shell.ExecutecmdLine, noWaiting)
visible = vbFalse
'This means: Do not display the UI of the started process
NwEngine.Shell.ExecuteEx(cmdLine, noWaiting, visible)
```

5.5.2.2 Customizing Your SAP GUI Packages - Examples

Example: Copying a customized `services` file

```
'This script can be added to the “On Installation End” section of a SAP GUI 710
package event script.
'It distributes your special version of the services file
'given these files exist inside a directory “CustomerFiles” on your installation
server share.
NwEngine.Context.Log.Write "Event: Copying customized services file"
If NwEngine.Shell.FileExist("%SAPSrcDir%\CustomerFiles\services") Then
     NwEngine.Shell.CopyFile "%SAPSrcDir%\CustomerFiles\services", "%WinSysDir%
\drivers\etc\services"
End If
```

Example: Setting registry values to configure auto-update settings for SAP Logon

```
'This script can be added to the “On Installation End” section of a
'SAP GUI 710 package event script to configure the auto update settings for SAP
Logon.
NwEngine.Context.Log.Write "Event: Setting the auto update registry key for SAP
Logon"strRegUpdate =_
 "HKLM\SOFTWARE\SAP\SAPsetup\SAPstart\AutoUpdate\SAPLogon.exe\UpdateMode"
strRegProb = "HKLM\SOFTWARE\SAP\SAPsetup\SAPstart\AutoUpdate\SAPLogon.exe\Prob"
'Option 1: Update Mode is switched on with update frequency = 10. The user is
not allowed to configure.'
NwEngine.Shell.SetRegValue strRegUpdate, "REG_SZ", "ForceOn"
NwEngine.Shell.SetRegValue strRegProb, "REG_DWORD", "10"
'Option 2: Update Mode is switched off. The user is not allowed to switch it on.'
NwEngine.Shell.SetRegValue strRegUpdate, "REG_SZ", "ForceOff"
'Option 3: Update Mode is switched on with update frequency = 10. The user is
allowed
'to change the configuration.'
NwEngine.Shell.SetRegValue strRegUpdate, "REG_SZ", "On"
NwEngine.Shell.SetRegValue strRegProb, "REG_DWORD", "10"
'Option 4: Update Mode is switched off. The user is allowed to change the
configuration.'
NwEngine.Shell.SetRegValue strRegUpdate, "REG_SZ", "Off"
'The same settings can be applied to the registry values for 'SAPLgPad.exe':
' strRegUpdate =
"HKLM\SOFTWARE\SAP\SAPsetup\SAPstart\AutoUpdate\SAPLgPad.exe\UpdateMode"'_
 strRegProb = "HKLM\SOFTWARE\SAP\SAPsetup\SAPstart\AutoUpdate\SAPLgPad.exe\Prob"
```

Example: Modifying the `services` file during installation

```
'This script can be added to the "On Installation End" section
of a SAP GUI 710 package event script.
```

```
'It checks whether a line containing the string "Alpha" exists.
'If so, it replaces that line with "Alpha 1901/tcp"
'Otherwise it simply appends the new line to the end of the services file.
NwEngine.Context.Log.Write "Event: Appending or replacing _
lines in the services file"
strFile = NwEngine.Variables.ResolveString(_
"%WinSysDir%\drivers\etc\services" )
Set objTextFile = CreateObject("NwSapSetupATLCommon.TextFileParser")
If objTextFile.Parse( strFile ) Then
    NwEngine.Context.Log.Write "Event: Modify the file " & Chr(34)
    & strFile & Chr(34)
    If objTextFile. Does String Exist ( "Alpha" ) Then
        objTextFile.ReplaceLineEx  "Alpha",  "Alpha 1901/tcp"
    Else
        objTextFile.AppendLine "Alpha 1901/tcp"
    End If
    objTextFile.Save( strFile )
Else
    NWEngine.Context.Log.WriteWarning "Event: Could not open the file "_
    & Chr(34) & strSalFile & Chr(34)
End If
```

Example: Modifying the `services` file after uninstallation

```
' This script can be added to the "On Installation End" section
' of a SAP GUI 710 package event script.
' It removes a line from the services file which was set in the last Example #4.
NwEngine.Context.Log.Write "Event: Removing a line from the services file"
strFile = NwEngine.Variables.ResolveString(_
"%WinSysDir%\drivers\etc\services" )
Set objTextFile = CreateObject("NwSapSetupATLCommon.TextFileParser")
If objTextFile.Parse( strFile ) Then
    NwEngine.Context.Log.Write "Event: Modify the file " & Chr(34)_
    & strFile & Chr(34)
    objTextFile.RemoveLine "Alpha 1901/tcp"
    objTextFile.Save( strFile )
Else
    NWEngine.Context.Log.WriteWarning "Event: Could not open the file "_
    & Chr(34) & strSalFile & Chr(34)
End If
```

5.5.3 Changing the Package Content

You can modify the package content by adding or removing SAP front end components to be installed with the package. You can also add event scripts that are executed before or after the update process.

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of your installation server.
2. Select the *Package Configuration* tab page.
   The packages are displayed on the left side of the screen.

3. Select the package whose content you want to update and choose *Change Package Content*.

   You can change the selection of SAP front end components. The added components are installed on the workstations. Removed components are uninstalled from the workstations if they are not part of any other package marked for installation. When updating a package, the package parameters are also refreshed. They now include new variables and no longer include those that belong to SAP front end components that are no longer part of this package.
4. **Optional:** Add event scripts to perform custom actions on the user's workstation during the update of the package. You can insert script samples delivered by SAP choosing *Insert Script Sample* and adapt them to your requirements.

   The scripts are executed before or after the update:
   - *On Update Start*: executed before the update of a package
   - *On Update End*: executed after the update of a package
5. Choose *Save*.

Results

Saving the package increases the version number and the package is marked for update on the workstation. When the package installation is updated on the workstation, the package components that have been added or removed by the administrator are automatically installed or uninstalled.

> ⓘ Note
>
> If you updated package event scripts or files copied by event scripts, choose *Mark for Update* on the *Configure Packages* tab to increase the package version. Choose *Save*. The installer now recognizes the package as updated, and the modifications are transferred to the workstations during the update with the event scripts.

5.5.4 Deleting an Installation Package

This section describes how to remove and delete packages permanently.

Prerequisites

> ⚠ Caution
>
> The package and its event scripts are permanently deleted without an option for recovery.

- An installation server is available.
- Packages are available.

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of your installation server.
2. On the *Packages* tab page, select the package to be deleted and choose *Delete Package* from the context menu.

Results

The package is deleted and removed from the list on the *Packages* tab page.

5.5.5 Creating and Deploying a Single-File Installer for Packages

The single-file installer for a package contains only those files of the SAP front end components that are a part of the package. This reduces the network load if you've to copy the distribution medium to the workstation before installing.

Prerequisites

- An installation server is available.
- Packages are available.
- If you want to distribute configuration files like the `services` file together with the single-file installer, create a directory named `CustomerFiles` in the root directory of your installation server and copy your files into this directory. All files and directories in this directory will be included in every package single-file installer. Use package event scripts to install your custom files.
  For more information, see Package Event Script Samples [page 22].

> → Tip
>
> The single-file installer can be executed on any workstation. To run the single-file installer silently without user interaction, enter **`/silent`** in the command line. More information: Command Line Parameters for the Installation Server [page 38].

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of your installation server.
2. On the *Packages* tab page, select the package for which you want to create a single-file installer and choose *Compress to Single-File Installer* from the context menu.

3. Follow the wizard instructions.
4. To deploy the single-file installer, follow the wizard instructions.

Results

You've created and deployed a single-file installer.

5.5.6 Creating a Package Definition File

This section describes how to create a package definition file (PDF) for an installation server package.

Context

PDF is a package description format that simplifies the interoperability and information exchange between SAPSetup and system management products such as Microsoft's Systems Management Server.

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of your installation server.
2. On the *Packages* tab page, select the package for which you want to create a PDF file and choose *Create Package Definition File* from the context menu.
3. In the upcoming window, choose the directory where you want to save the file, enter a file name, and save your entries.

Results

`NwSapSetupAdmin.exe` creates a PDF and a Microsoft Systems Management Server (SMS) file in the specified directory.

5.6 Patching the Installation Server

Patching SAP front end components on the installation server keeps them up to date with the most recent correction and enhancements from SAP.

Context

> → Tip
>
> You can configure the Automatic Workstation Update Service (AWUS). This service updates the workstations and reboots them whenever the installation server is patched or the installed packages are updated, if necessary. The AWUS also works if no user is logged on to the workstation.
>
> More information: Installing and Configuring the Automatic Workstation Update Service [page 32]

> ⓘ Note
>
> To ensure a successful patch and to prevent having to reboot after the patch, you can unshare the installation server during the patch and share it again when the patch has finished.

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of the installation server.
2. Choose *Patch Server*.
   This starts the patch wizard.
3. To continue, choose *Next*.
4. Browse to the SAP patch file and choose *Next*.

   `NwSapSetupAdmin.exe` validates the patch. If the patch is valid, you are prompted to proceed.

   > ⓘ Note
   >
   > When the patch process starts, `NwSapSetupAdmin.exe` is closed. It restarts after the patch process has completed. This ensures that the installation server is not modified during the patch process.

5. To continue, choose *Next*.
   `NwUpdateInstServer.exe` starts patching the installation server. A log file is created in the SAPSetup log file directory.
6. Follow the wizard instructions.

   The wizard confirms the successful patch process. If an error occurs, see Troubleshooting [page 51].
7. Choose *Close*.
   `NwSapSetupAdmin.exe` now restarts.

Results

Patching a SAP front end component (for example, SAP GUI for Windows 7.30) increases the version of each package that contains this component.

5.7 Configuring the Local Security Handling

Local security handling (LSH) enables users to install SAP front end components on their workstations without requiring administrator privileges. By configuring LSH, the distribution service is installed and started.

Prerequisites

- The installation server needs to be available and accessible via file sharing from the workstation.
- Server and workstations must be part of a Windows domain or active directory.
- Two domain user accounts are needed to run the installation service and the distribution service. The user for the distribution service must have administrative privileges on each workstation. The user for the installation service will get administrative privileges on the workstation temporarily, if he doesn't have them anyway. You can use the same user account for both services.
- It must be possible to control services via the remote service management on the workstations. Make sure to configure the firewall accordingly.

> → Remember
>
> To enable workstations that run Microsoft Windows Vista to use LSH, you have to change the default firewall setting for remote service management:
>
> 1. Start the Control Panel.
> 2. Choose *Windows Firewall* > *Change Settings*.
> 3. Select the *Exceptions* tab page.
> 4. Select *Remote Service Management* and choose *OK*.

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of the installation server.

   You need administrator privileges on the installation server.
2. Choose *Services* and select *Maintain Local Security Handling* > *Configure Local Security Handling*.
   The LSH configuration wizard starts.
3. Choose *Next*.
   The wizard prompts you for the credentials of an account that has administrator privileges on all workstations on the network. The distribution service will run with these credentials.

4. Enter the account name with domain qualifiers like `domain\user`, enter the password twice, and choose *Verify*.

   Credentials are **not** validated. The verification only confirms that the password supplied matches its repetition.
5. Choose *Next*.
6. Enter the details for the installation service account and choose *Verify*.
7. To complete the process, choose *Next*.
   The wizard indicates whether LSH has been configured successfully. In this case the state of the distribution service is displayed as *Active* in the status bar at the bottom of `NwSapSetupAdmin.exe`.
8. Test the LSH on a workstation as a user without administrator privileges by starting `NwSapSetup.exe` from this installation server.

Results

The configuration is successful if the front end installer starts and is able to install SAP front end components which are available on the installation server. The status bar shows the state of the distribution service.

If the configuration was not successful, you have to reconfigure LSH. For more information, see 1162270.

> Note
>
> - To **temporarily** disable LSH, choose *Services* > *Maintain Local Security Handling* > *Stop Distribution Service*. Enable LSH again by choosing *Start Distribution Service* in the same menu.
> - To **permanently** disable LSH, choose *Services* > *Maintain Local Security Handling* > *Uninstall Distribution Service*.
>
> If the service is not configured, these menu entries are disabled.

5.8 Installing and Configuring the Automatic Workstation Update Service

If the automatic workstation update service (AWUS) is configured, the service updates the workstation and reboots it, if required. This happens whenever the installation server is updated or patched, or installed packages are updated.

Prerequisites

- The installation server was created and configured with `NwCreateInstServer.exe`.
- The installation server should be hosted on a machine that can work as a file server and serve numerous network sessions.

- Windows server as operating system with the following local security policy:
  - `Accounts: Guest account status` = Enabled
  - `Network access: Let ‘Everyone’ permissions apply to anonymous users` = Enabled
  - Share is `NULL-session accessible`
- The workstation needs network access to the installation server.

Context

The AWUS is shipped with most SAP front end products and works only when installed on workstation as decribed below. During installation, the installation server from where the installation has been started is stored as source location for updates. The service is updated automatically whenever a patch is available. The AWUS works regardless of whether a user is logged on or not.

- If a user is logged on, he is informed of the availability of an update. The update starts only on the user's assent. The user is also informed about whether a reboot is required, and the reboot is executed only upon confirmation by the user.
- If no user is logged on, the update and the reboot (if necessary) are started automatically.

Procedure

1. Configure the AWUS:
   a. Start `NwSapSetupAdmin.exe` from the `Setup` directory of the installation server.
   b. Choose *Services* and select *Configure Autmatic Workstation Update*.

      The dialog for configuring the AWUS is displayed.
   c. Define the AWUS configuration settings:

Option	Result
Update check frequency	The workstation will check for updates at this interval. The default value is 24 hours.
Enforce reboot	With these options, you can control whether a reboot of the workstation should be enforced after an update. *Enforce reboot if needed (recommended)*: A reboot will take place, if it is necessary to complete the update. *Enforce reboot after every update*: A reboot will be initiated after every update. *Don't enforce reboot* : No reboot will be initiated. In case a reboot is initiated and a user is logged on, the user will be notified. Reboot will only be performed after getting his approval.

Option	Result
Additional update sources	You can list additional installation servers. The AWUS will check the listed installation servers for available updates in the given order. By default, the AWUS will always check the installation server from which it had been installed itself.
Disable automatic workstation updates	The AWUS on the workstation is disabled the next time it checks for updates. It will not execute updates from this server, until you enable it again. To permanently stop AWUS on a workstation, uninstall it from the workstation.

d. Save your entries and close the dialog.

2. Install the AWUS on the workstation:
   a. Create a package containing the component `SAP Automatic Workstation Update`.
      More information: Creating an Installation Package [page 20]
   b. Deploy the package on the workstation.
      More information: Installing Packages Configured by the Administrator [page 41]

   Once the AWUS is installed, the following programs will run on the workstation in background mode:

   - `NwSapSetupUserNotificationTool.exe`
   - `NwSapAutoWorkstationUpdateService.exe`

> **Note**
>
> The AWUS checks for updates on the last 10 installation sources that are network paths.

Results

You have configured the AWUS and installed it on the workstation.

5.9 Controlling a Remote Workstation

With the remote workstation control, you can access and control workstations remotely on which you've administrator privileges.

Prerequisites

- Check if the installation server is available.
- Enable the Windows Management Instrumentation (WMI).
- Check that WMI access isn't blocked by the firewall.

- Check if you've either domain administrator privileges or local administrator privileges on the workstation.

> → Tip
>
> Use the Microsoft tool `wbemtest.exe` to check the WMI connectivity to the remote workstation. Enter the connection namespace as **`\\<WorkstationName>\root\cimv2`**.

Features

The remote workstation control helps you with the following tasks:

- Collecting log files from workstations.
  More information: Collecting Log Files Remotely [page 35]
- Running a process on the workstation.
  More information: Executing a Process Remotely [page 36]
- Displaying a list of processes running on the workstation.
  More information: Listing Remote Processes [page 37]

5.9.1 Collecting Log Files Remotely

This section describes how to collect log files from a workstation remotely.

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of the installation server.
2. Choose *Remote* > *Collect Log Files (WMI)*.
3. Enter the name of the workstation. If you want to collect log files from several workstations, separate the host names with a comma.

   > → Tip
   >
   > If your current user has no administrator privileges on the remote workstation, enter the credentials of a user with administrator privileges on the remote workstation in screen area *Administrative Credentials*. This data will not be saved.

4. Choose *Collect Logs* to start the collection of log files from the remote workstation.

   > ⓘ Note
   >
   > If you want to stop the collection process, choose *Abort*.

   The files from the workstation are copied to directory `%temp% \SAPRemoteWksta`. Once the file collection has completed, the directory is opened.

5.9.2 Executing a Process Remotely

This section describes how to execute a process on the workstation remotely.

Prerequisites

- The installation server has to be shared.
- `NwSapSetupAdmin.exe` has to be started as a user that is local administrator on the remote workstation on which you want to execute the process.
- If you want to start an installation on the remote workstation from the installation server, you have to activate the delegation function for the workstation. For more information, see the Microsoft TechNet article , question #10.

Context

You can execute installs, updates, repairs, and uninstalls remotely on a single remote workstation. `NwSapSetup.exe` is started from the installation server and runs on a remote computer. This is useful to provide quick help to a user in trouble, or to deploy new software for tests on a small number of computers.

`NwSapSetup.exe` will be installed as a Windows service that removes itself after the process is complete. The Windows service will always be removed, even if an installation error occurs, so that no artefact is left.

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of the installation server.
2. Choose *Remote* > *Execute on Remote Workstation*.

   The *Execute process on a remote computer* dialog is displayed.
3. Choose the action that you want to execute:
   - *Install*: will install any selected product or package that is not yet installed on the remote workstation, and it will update any selected product or package that is already installed on the remote workstation.
   - *Update*: will enable the item *All*, so that you can update all installed products on the remote computer with the version on the installation server. If you select a product or package that is not yet installed on the remote computer, it will be installed. No downgrade will be done.
   - *Repair*: will repair installed SAP front end components [page 46] on the remote computer.
   - *Uninstall*: will uninstall the selected SAP front end components from the remote workstation.
4. Choose the item that you want to be processed.

   The available items depend on the action you chose. The options *Install All* and *Repair Product* are not available.

> **Note**
>
> Repairing a package is useful to execute your package event scripts [page 22] again. Make sure to increase the version of the package beforehand.

The command line [page 47] that will get executed is displayed at the bottom of the dialog, you can copy it.

5. Enter the name of the remote computer you want to execute the process on.
6. Choose if you want to execute the process as user *LocalSystem* or as a different user.

   If your installation server is shared as a null session share, the process will run as user *LocalSystem*. Otherwise you have to specify the credentials of a user that is local administrator on the remote workstation and has access to the shared installation server.

   More information: Microsoft support article about null sesstion share enablement
7. Choose *Execute* to run the process on the remote machine.

   After choosing *Execute*, `NwSapSetupAdmin` waits for the remote process to start. A message box informs if the start was successful. There is no feedback on the progress of the process, nor when it is finished. It runs now independently from your actions. You can leave the dialog or start the next remote process.
8. To check if the process has finished, choose *Remote* > *Remote Task Manager (WMI)* and check for the process [page 37] `NwSapSetup.exe`.
9. To check the results, choose *Remote* > *Collect Log Files (WMI)* and check the return code [page 53] of the latest `NwSapSetup.log` file.

> **Note**
>
> If you encounter connection issues, see 1162270.

5.9.3 Listing Remote Processes

This section describes how to display the list of processes running on the workstation.

Procedure

1. Start `NwSapSetupAdmin.exe` from the `Setup` directory of the installation server.
2. Choose *Remote* > *Remote Task Manager (WMI)*.
3. Enter the name of the workstation and choose *Display*.

> **Note**
>
> If you want to terminate a process, select it and choose *Terminate*. Be aware that in this case, the user who is working on the remote machine might lose unsaved data.

5.10 Removing the Installation Server

This section describes how to remove an installation server if it is no longer required.

Procedure

1. If the LSH [page 31] is configured, start `NwSapSetupAdmin.exe` and choose *Services* > *Stop Distribution Service*.
2. Unshare the directory of the installation server.
   As a result the installation server is unavailable for workstations.
3. Delete the directory containing the installation server.

5.11 Command Line Parameters for the Installation Server

This section provides command line parameters for installation servers. They are not case-sensitive.

Parameters for *NwCreateInstServer.exe* and *NwUpdateInstServer.exe*

Parameter	Description
`/dest`	Destination directory where you want to create your installation server. Example **`/dest="C:\MyInstServerPath" /silent`**
`/nodlg`	Shows only the progress dialog. Displays no other user interface. You can use it instead of **`/silent`**, if you wish to see the progress. Note If you use **`/nodlg`**, you have to supply the destination path with **`/dest:<destination path>`**

Parameter	Description
`/silent`	Does not display a user interface. **Note** If you use `/silent`, you have to supply the destination path with `/dest:<destination path>`
`/DontConfigureServerPath`	Disables the automatic configuration of the installation source directory (network share creation and null-session accessibility) when creating an installation server with the command line.

Parameter for *NwSapSetupAdmin.exe*

Parameter	Description
`/checkserver`	Verifies the integrity of the installation server in silent mode (see Troubleshooting [page 51]). Returns an error level greater than zero and writes an error file, if discrepancies are found (see Log and Error Files [page 52]).

6 Installing and Administering SAP Front End Components and Packages

This section describes how to install, update, repair, and remove SAP front end components and packages on a workstation.

6.1 Installing SAP Front End Components

This section describes how to install SAP front end components from an installation server or a distribution medium.

Prerequisites

SAP front end software applications should not be running during installation. If an application is running, the user may be prompted to reboot afterwards to complete the installation. If the installation runs interactively, the user is prompted to stop the applications. If he does so, no reboot is necessary.

Procedure

1. Choose one of the following scenarios:
   - **Installing SAP front end components from an installation server**
     1. Check if the user logged on to the workstation has local administrator privileges. If not, LSH has to be configured on the installation server.
        More information: Configuring the Local Security Handling [page 31].
     2. Start `NwSapSetup.exe` from the `Setup` directory on the installation server.
   - **Installing SAP front end components on a single workstation from a distribution medium**
     1. Check if the user logged on to the workstation has local administrator privileges.
     2. Start `NwSapSetup.exe` from your installation medium.

        > ✤ Example
        >
        > On the *SAP NetWeaver Presentation DVD*, the setup executable is located at `GUI\WINDOWS\WIN32`.

   The SAPSetup installation wizard starts and displays a list of components that are part of the product to be installed.

> → Tip
>
> Starting `SetupAll.exe`, you can install all SAP front end components available on the installation server or distribution medium in one run.

2. On the SAPSetup installation wizard, choose *Next*.
   The list of SAP front end component is displayed. Already installed components are preselected.
3. Select the SAP front end components that you want to install and then choose *Next*.

   A green plus sign indicates that this SAP front end component will be installed on the workstation, or that it is aready installed and will be updated.

   > ⓘ Note
   >
   > Changes in the selection list are indicated by yellow dots.

4. If some products ask for additional parameters during the installation, like the installation directory, enter the required information and then choose *Next*,.

Results

Your SAP front end components are now successfully installed and ready to use on the workstation. Choose *Finish* to exit the installer.

> ⓘ Note
>
> Processing is recorded in the SAPSetup log file. For more information, see Troubleshooting [page 51] and Log and Error Files [page 52].

6.2 Installing Packages Configured by the Administrator

This section describes how to install packages on the installation server that have been configured by the administrator.

Prerequisites

Check if the user logged on to the workstation has local administrator privileges. If not, LSH has to be configured on the installation server.

More information: Configuring the Local Security Handling [page 31]

Procedure

1. Start `NwSapSetup.exe` from the `Setup` directory on the installation server or distribution medium, choose *Next* on the welcome screen and then *Predefined Packages*. Alternatively, call `NwSapSetup.exe /package` on the command line, and choose *Next* on the welcome screen.
   You see all packages of the installation server in a list. Packages that have already been installed are preselected.
2. Select the packages that you want to install.

   A green plus sign indicates that this package will be installed on the workstation. A yellow dot indicates a change in the selection list.
3. Choose *Next*.
   `SAPSetup` now processes the packages and displays the completion status when finished.

Results

Your packages are now successfully installed and ready to use.

> ⓘ Note
>
> Processing is recorded in the SAPSetup log file. For more information, see Troubleshooting [page 51] and Log and Error Files [page 52].

6.3 Installing Using the Windows Logon Script

This section describes how to distribute SAP Front End components and packages on many workstations in an automated way. Inserting a command line into the logon script of the users, the installation is started when the user logs on.

Prerequisites

Check that SAP front end components (especially SAP GUI) are closed before you start the task. SAPSetup might require a reboot to complete the installation if program files are in use.

Context

When you start the command on a workstation for the first time, SAPSetup installs the specified package. SAPSetup will check for package updates on the installation server every following logon, and will install updates automatically.

> **Note**
>
> You can find the command line name of a package or SAP front end component in the administrative console `NwSapSetupAdmin.exe`. For more information on options, see Command Line Parameters for the Workstation [page 47] and Return Codes [page 53].

Procedure

- To install SAP front end components, add the following command to the user's logon script:

```
\\<server>\<shared directory>\Setup\NwSapSetup.exe
         /product:"<product command line name>" /nodlg
```

- To install packages, add the following command to the user's logon script:

```
\\<server>\<shared directory>\Setup\NwSapSetup.exe
        /package:"<package command line name>" /nodlg
```

- **Optional:** To ensure that an installation runs only once on a workstation, add the following option: `/once:"<OnceTag>"`

  The tag `<OnceTag>` is stored in the Windows registry and is never deleted by SAPSetup. If an installation with the same tag is started again, it does nothing and instantly returns 0. This can be useful for distributing a correction to many workstations using event scripts in packages that contain no SAP front end components.

6.4 Updating SAP Front End Components

This section describes how to update the installed SAP front end components when a newer version is available as a patch either on the installation server or on a distribution medium.

Prerequisites

Check that SAP front end components are closed before you start the task. SAPSetup might require a reboot to complete the update if program files are in use.

Context

This procedure updates all products that have been installed without using packages.

> **Note**
>
> If you make use of packages, call this command with adding the parameter `/package`. If you distribute software both with and without packages, call this command twice, once with the parameter `/package` and once without.

Procedure

- **Updating a Workstation from an Installation Server**

  Updating SAP front end components on a workstation involves the same steps as an installation [page 40]. You can choose to run the update unattended or automatically.

  - For an **unattended update**, use the following command line: `\\<server_path>\Setup\NwSapSetup.exe /update /silent`.

    > **Note**
    >
    > If you want to update a specific package or SAP front end component, add `/package="package command line name"` or `/product="product command line name"`. You can find the command line name of the package or SAP front end component in the administrative console `NwSapSetupAdmin.exe`.

  - For an **automatic update**, you use the AWUS [page 32]. Whenever the installation server is patched or the installed packages are updated, this service updates the workstations and reboots them, if necessary. The AWUS works regardless of whether a user is logged on.
    - If a user is logged on, the user is informed of the available update, and the update starts upon confirmation by the user. The user is also informed about whether a reboot is required, and the reboot is executed only upon confirmation by the user.
    - If no user is logged on, the update and the reboot (if necessary) are started automatically.

  Depending on the chosen update method, the SAP front end components are updated with or without user interaction. If the interactive update method is selected, the user simply has to choose *Next* to start the front end update.

- **Updating a Standalone Workstation**

  We recommend you to update a standalone workstation from an installation server [page 44] by means of the automatic update mechanisms (AWUS or logon scripts) which ensures a fast and efficient update of all workstations. However, you can update SAP front end components on standalone workstations by running a patch on every single workstation. In this case, an update involves the same steps as an installation [page 40].

6.5 Uninstalling SAP Front End Components

This section describes how to remove SAP front end components from a workstation.

Prerequisites

SAP front end software applications should not be running during uninstallation. If an application is running, the user gets a message to reboot the workstation.

> ⚠ Caution
>
> You cannot remove a SAP front end component with a SAPSetup version less than 8.6 if the component was installed with version 8.6 or higher. This will lead to an intentional termination of SAPSetup.

Procedure

1. On the workstation, choose *Start* > *Settings* > *Control Panel* > *Add or Remove Programs*.

   A list of the installed applications is displayed.
2. Select the SAP front end components to be removed and follow the instructions of the wizard.

   If you want to perform a silent or unattended uninstallation, enter the following commands:
   - For silently uninstalling a certain product: `NwSapSetup.exe /Product="<product name>" /Silent /Uninstall`
   - For silently uninstalling all SAP front end components installed by SAPSetup: `NwSapSetup.exe /all /silent /uninstall`
   - For an unattended uninstallation of all SAP front end components installed by SAPSetup: `NwSapSetup.exe /all /nodlg /uninstall`

   When the command line options include `/uninstall /all`, the event scripts defined for the events `On Uninstallation Start` and `On Uninstallation End` will also be executed for all packages that are being uninstalled.

   For more command line options, see Command Line Parameters for the Workstation [page 47].

Results

The selected SAP front end components are removed from the workstation.

> ⓘ Note
>
> Processing is recorded in the SAPSetup log file. For more information, see Troubleshooting [page 51] and Log and Error Files [page 52].

6.6 Repairing Installed SAP Front End Components

This section describes how to repair SAP front end components which are installed on a worksation and do not work properly.

Prerequisites

- Check if the user logged on to the workstation has local administrator privileges. If not, LSH has to be configured on the installation server.
  More information: Configuring the Local Security Handling [page 31]
- SAP front end software applications should not be running during a repair. If an application is running, the user might get a message to reboot the workstation. The repair process is completed after restart.

Context

If SAP front end components on a workstation are not working properly, start the repair process. This process checks for discrepancies in files, services, registry-keys and other artifacts installed by SAPSetup, and performs a re-install of differing artifacts. The repair process includes an update of SAP front end components on the workstation with versions lower than on the installation server. If a package is installed, the package event scripts [page 21] for updates are also executed during the repair.

Procedure

- To start the repair process, you execute program `NwSapSetup` with parameter `repair` from the `Setup` directory either of the installation server, the distribution medium, or a single-file installer:
  ```
  NwSapSetup.exe /repair
  ```

  The repair process starts and a progress screen is displayed. Only those SAP front end components that are available on the source can be repaired. The processing is recorded in the file `NwSapSetup.log` in the SAPSetup log file directory.

  The repair process includes an update of all SAP front end components. If the version of a component on the installation server is higher than that of the component installed on the workstation, this component is udpated.

  If a package is installed, the package event scripts [page 21] for updates are executed during repair.

  > **Note**
  >
  > If errors occur, see Troubleshooting [page 51].

Results

Your SAP front end components are now successfully repaired and ready to use on the workstation.

6.7 Command Line Parameters for the Workstation

This section provides command line parameters for the workstation.

When using the command line, consider the following rules:

- Parameter names and values are not case-sensitive.
- Specify parameter names either by `/param` or `-param`.
- Separate parameter name and value either by `:` or `=`.
- Surround parameter values that contain spaces either by `"` or `'`.
- Some Parameters can take multiple values. Separate them either by `+` or `,`.

> ✤ Example
>
> ```
> NwSapSetup.exe /uninstall /all
> NwSapSetup.exe -repair - package="myPackage"
> NwSapSetup.exe -product:SAPGUI710
> NwSapSetup.exe /uninstall /package:'my sap gui package,my nwbc package'
> ```

Parameters for *NwSapSetup.exe*

Parameter	Impact
`/silent`	SAPSetup does not display a user interface. A splash screen is also not shown. > ⓘ Note > When using `/silent`, you have to specify a product name, a package name, or `/update`.
`/nodlg`	SAPSetup only shows the progress dialog. It does not display any other user interface. You can use it instead of `/silent`. > ⓘ Note > When using `/nodlg`, you have to specify a product name, a package name, or `/update`.
`/nosplash`	SAPSetup does not display a splash screen at startup.

Parameter	Impact
`/force`	SAPSetup overwrites all files, registry keys, and other artifacts installed by earlier runs of SAPSetup, regardless whether they exist or not. Even files with a newer file version are overwritten. When used for uninstall, SAP files are uninstalled even if their shared DLL counter is not zero. Superdeeded by `/repair`.
`/once:"<OnceTag>"`	Makes sure that an installation with the `<OnceTag>` tag runs exactly once on a workstation. The `<OnceTag>` tag is stored in the Windows registry and never deleted by SAPSetup. If an installation with the same tag is started again, it will do nothing and return 0 instantly.
`/uninstall`	Uninstalls SAP front end components installed by SAPSetup. • To uninstall all SAP front end components, add `/all`. • To uninstall a specific product or package instead, add `/product="product command line name"` or respectively `/package="package command line name"` **Note** Works only with `/nodlg` or `/silent`.
`/product`	SAPSetup runs the installer in product mode. You cannot switch to *Package View*.
`/package`	SAPSetup runs the installer in package mode. You cannot switch to *Product View*.
`/product:"<product cmd name>"`	Only the specified product is processed. Other products are not displayed on the selection dialog. You can specify several products by concatenating their names with a plus sign (+) or a comma (,). Ignored when `/repair` is specified, as due to possible dependencies only all products can be repaired .
`/package:"<package cmd name>"`	Only the specified package is processed. Other packages are not displayed on the selection dialog. You can specify several packages by concatenating their names with a plus sign (+) or a comma (,). When `/repair` is specified, only the specified packages are repaired.

Parameter	Impact
`/update`	Updates installed SAP front end components. • To update all installed packages that are available on the installation server, add `/Package`. • To update a specific product or package, add `/product="product command line name"` or respectively `/package="package command line name"`. **Note** Without specifying `/package`, packages are ignored. **Note** Works only with `/nodlg` or `/silent`.
`/skip=wtscheck`	Skips the check whether the WTS server is in installation mode. Only provide this if the check does not work correctly. In this case, open a message on component BC-FES-INS for a bug report.
`/ForceWindowsRestart`	Forces a restart of the workstation after the installation is complete. Use in combination with `/package` or `/product` after `/silent` or `/nodlg`. Example: `\NwSapSetup.exe /silent /product="SAPGUI" /ForceWindowsRestart`
`/SMS[:"<package cmd name>"]`	Creates a status file `<package command line name>.MIF` in the `%TEMP%` directory that indicates the success or failure of the process. Typically used by software distribution systems such as SMS to determine the success or failure of a remote installation (advertised package installation).
`/repair`	Repairs all installed SAP front end components. When combined with `/package:"pckg"`, only the specified package is repaired. Ignores products specified with `/product:"p"`. More information: Repairing Installed SAP Front End Components [page 46]
`/MaintenanceMode`	Usually, only those SAP front end components that are also available on the installation source are displayed in the selection dialog. With this switch, all installed SAP front end components are displayed, even if they are not available on the installation source, and can be uninstalled.

Parameter for *Single-File Installer*

Parameter	Description
`/?`	Provides help on command line parameters.
`/extract:dest_dir`	Extracts the single-file installer into the `dest_dir` directory. If the directory does not exist, it will be created. This works with single-file installers shipped by SAP as well as with those created in NwSapSetupAdmin [page 28].

Parameter	Description
`/CreateServer [/dest="dest_dir"]`	Creates a new installation server. The optional parameter `/dest`specifies the target directory of the installation server. This directory must exist and should be empty. This call is equivalent to extracting the single-file installer and then manually starting `Setup\NwCreateInstServer.exe`.
`/UpdateServer [/dest="dest_dir"]`	Updates an existing installation server. The optional parameter `/dest`specifies the target directory of the installation server that will be updated. This directory must exist. This call is equivalent to extracting the single-file installer and then manually starting `Setup\NwUpdateInstServer.exe`.
`/repair`	Repairs all installed SAP front end components. More information: Repairing Installed SAP Front End Components [page 46]
`/silent`	Installs all SAP front end components contained in the single file installer without displaying a user interface. Installed SAP front end components will be updated, if the single-file installer contains a newer version. Compatible with all other parameters except `/?`.
`/nodlg`	Installs all SAP front end components contained in the single file installer showing only the progress dialog. Installed SAP front end components will be updated, if the single-file installer contains a newer version. Compatible with all other parameters except `/?`.

Parameter for *NWCheckWorkstation.exe*

Parameter	Description
`/silent`	Does not display a user interface. Returns the same return values as `NwSapSetup.exe`. For more information, see Return Codes [page 53].

7 Troubleshooting

This section describes how to check the integrity of the installation server and the installed SAP front end components on a workstation.

Procedure

- **Troubleshooting on the Installation Server**
  - Integrity check
    1. Start `NwSapSetupAdmin.exe` from the `Setup` directory.
    2. Choose *Check Server* and follow the wizard instructions.
       This checks whether all required files are available and unchanged. If files are missing or if their hash values are incorrect, the wizard displays a link to an error report.
    3. To view the error report in a web browser, choose the provided link. Otherwise, choose *Close* to exit the wizard.

    This integrity check is helpful if you face problems with creating a single-file installer, for example.
    If the integrity check reveals incorrect hash values, your server may be corrupt. You can have the hashes recalculated by choosing *Repair Metadata* in `NwSapSetupAdmin.exe` only if you yourself have changed the files on the installation server. For more information, see Log and Error Files [page 52].
  - On the *Packages* tab, the packages are displayed in the navigation pane on the left side of the screen.

    The icon indicates that the package is incomplete. This occurs when you deleted a SAP front end component contained in the package from the installation server. In this case, delete the package, reimport the missing SAP front end component, or change the package content.
- **Troubleshooting on the Workstation**

  `NwCheckWorkstation.exe` verifies the installation of SAP front end components by checking for discrepancies in files, services, registry-keys, and other artifacts installed by SAPSetup. You do not require administrator privileges to perform workstation checks. If there are discrepancies, a report is displayed.

  The workstation check tool collects installation data and log files, and compresses them into a cabinet archive. After the check is completed, Windows Explorer opens the directory containing this CAB file. When reporting installation issues, you should forward the CAB file to SAP support for a quicker diagnosis.

  To start the workstation check, proceed as follows:

  1. Start `NwCheckWorkstation.exe` from the `Setup` directory either of the installation server or of the workstation (`%ProgramFiles%\SAP\SapSetup\Setup` on 32bit Windows or `%ProgramFiles32%\SAP\SapSetup\Setup` on 64bit Windows)
  2. Follow the wizard instructions.

  If an error occurs during processing, the wizard displays a link to an error report. Follow the link to view the error report in a web browser.

7.1 Log and Error Files

All the installation tools described in this document maintain an activity record in log files. Errors are saved in an XML format in files that you can view in your web browser.

Log files

The log files are stored in the following directory:

- Windows 32bit: `%ProgramFiles%\SAP\SapSetup\LOGs`
- Windows 64bit: `%ProgramFiles(x86)%\SAP\SapSetup\LOGs`

Each tool stores the last 20 log files.

> → Recommendation
>
> For technical reasons, the installation tools write various log files that among other things contain user names. In order to comply with data protection, privacy and security requirements, we strongly recommend that you delete these log files as soon as you no longer need them.

Error files

The error files are stored in the following directory:

- Windows 32bit: `%ProgramFiles%\SAP\SapSetup\Errors`
- Windows 64bit: `%ProgramFiles(x86)%\SAP\SapSetup\Errors`

Tool releated log and error files

Tool	Log File	Error File
`NwCreateInstServer.exe`	`NwCreateInstServer.log`	`NwCreateInstServerErrors_<DateTime>.xml`
`NwUpdateInstServer.exe`	`NwUpdateInstServer.log`	`NwUpdateInstServerErrors_<DateTime>.xml`
`NwSapSetupAdmin.exe`	`NwSapSetupAdmin.log`	`NwSapSetupAdminErrors_<DateTime>.xml`
`NwSapSetup.exe` single-file installer	`NwSapSetup.log`	`SAPSetupErrors_<DateTime>.xml`

Creating a message

Use component BC-FES-INS to create a message and attach the relevant log files and error files.

If you run `NwCheckWorkstation.exe` on the workstation as described in section Troubleshooting [page 51], attach the generated CAB file instead.

7.2 Return Codes

If you start `NwSapSetup.exe` from a batch file, the environment variable `%ERRORLEVEL%` contains the return code.

Example

Example

```
start /wait <Path to installer source>\Setup\NwSapSetup.exe /package="<cmd line name of your package>" /silent && echo %ERRORLEVEL%
```

The following table describes the return codes for `NwSapSetup.exe`:

Return Codes	Description
0	Process ended without detected errors
3	Another instance of SAPSetup is running
4	LSH failed
16	SAPSetup started on WTS without administrator privileges
26	WTS is not in install mode
27	An error occurred in COM
48	General error
67	Installation canceled by user
68	Invalid patch
69	Installation engine registration failed
70	The following reasons are possible: • A prerequisite for the installation was not met and the installation was executed with `/silent` or `/nodlg`. • Invalid XML files
129	Reboot is recommended

Return Codes	Description
130	Reboot was forced
144	Error report created
145	Error report created and reboot recommended
146	Error report created and reboot forced

> **Note**
>
> In event of return codes 144-146, check the Log and Error Files [page 52] for `NwSapSetup.exe`.

8 Scripting

SapSetup can be extended by scripts. It currently supports Visual Basic Script , but due to deprecation from Microsoft, a replacement is needed. For this, SapSetup can be extended with PowerShell script.

- PowerShell and NwEngine [page 55]
- Visual Basic Script Application Program Interfaces [page 102]

8.1 PowerShell and NwEngine

SapSetup provides PowerShell objects for easy scripting.

- **NwEngine.Shell [page 55]**
  Handling of files and directories, Windows registry entries, executing and controlling processes.
- **NwEngine.Context [page 83]**
  Checking the context of the installation, like Windows version, host name and command line arguments
- **NwEngine.Context.Log [page 90]**
  Writing log file entries
- **NwEngine.Variables [page 92]**
  Getting values of SapSetup variables

For more information on how to convert Visual Basic Scripts to PowerShell, check The VBScript-to-Windows PowerShell Conversion Guide .

8.1.1 NwEngine.Shell

`NwEngine.Shell` comprises methods to access and manipulate the file system and the Windows registry.

File System Access

Methods for File System Access

Method Name	Description
`CopyDirectoryEx` [page 58]	Copies source directory to target directory with all content. Returns `true` if copying was successful, else `false`.

Method Name	Description
`CopyDirectoryEx` [page 59]	Copies source directory to target directory with all content. If parameter `failIfExists` is set to `true`, existing files will not be overwritten. Returns `true` if copying was successful, else `false`.
`CopyFile` [page 60]	Copies source file to target file on the file system. Returns `true` if copying was successful, else `false`.
`CopyFileEx` [page 61]	Copies source file to target file on the file system. If parameter `failIfExists` is set to `true`, an existing file will not be overwritten. Returns `true` if copying was successful, else `false`.
`CreateDirectory` [page 61]	Creates a directory. Returns `true` if creation was successful, else `false`.
`CreateTempDirectory` [page 62]	Creates a temporary directory and returns the path as string.
`DeleteDirectory` [page 63]	Deletes a directory. Returns `true` if deletion was successful, else `false`.
`DeleteFile` [page 63]	Deletes a file. Returns `true` if deletion was successful, else `false`.
`DirectoryExist` [page 66]	Returns `true` if the directory exists, else `false`.
`FileExist` [page 71]	Returns `true` if the file exists, else `false`.
`GetDriveType` [page 71]	Returns a value indicating the type of the drive as integer.
`GetFileExt` [page 72]	Returns the extension of a file name as string.
`GetFileName` [page 72]	Returns the name of the path as string.
`GetFileSize` [page 73]	Returns the size of the file in bytes as long.
`GetFileVersion` [page 74]	Returns the version number of the file as string.
`GetPath` [page 74]	Returns the path of a file as string.
`GetShortPath` [page 76]	Returns the short path in 8.3 format of a file as string.
`IsPathEqual` [page 78]	Compares the paths of two directories and returns `true` if they are identical, else `false`.

Process and System Interaction

Methods for Process and System Interaction

Method Name	Description
`Execute` [page 69]	Executes a command line. Returns `true` if the process has finished, else `false`.
`ExecuteEx` [page 69]	Executes a command line, optionally without displaying a window. Returns `true` if the process has finished, else `false`.
`ExecuteRC` [page 70]	Executes a command line, optionally without displaying a window. Returns the return code of the process, if executed synchronously.
`GetSystemUtcTime` [page 77]	Returns the system UTC time as string.
`IsExeRunning` [page 77]	Returns `true` if the file is a running executable, else `false`.
`Sleep` [page 82]	Make SapSetup sleep for the specified time.
`TerminateProcessInstances` [page 82]	Terminates all processes whose executable name is given as parameter.

Registry Access

> **Note**
>
> On 64-bit Windows, SapSetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 83] property.

> **Note**
>
> To access the `(Default)` value of a registry key, just add a backslash `\` at the end of the registry key. To access the `(Default)` value from `HKLM\SOFTWARE\SAP`, add a backslash at the end: `HKLM\SOFTWARE\SAP\`.

Methods for Registry Access

Method Name	Description
`DeleteRegKey` [page 64]	Deletes a registry key and returns `true` if deletion was successful, else `false`.
`DeleteRegValue` [page 65]	Deletes a registry value and returns `true` if deletion was successful, else `false`.

Method Name	Description
`EnumRegKeys` [page 67]	Returns a collection containing all keys below the given path.
`EnumRegValues` [page 68]	Returns a collection containing all values below the given path.
`GetRegValue` [page 75]	Returns the data of the registry value.
`RegKeyExist` [page 78]	Returns `true` if registry key exists, else `false`.
`RegValueExist` [page 79]	Returns `true` if registry value exists, else `false`.
`SetRegValue` [page 80]	Sets a registry value.

Properties

Properties

Property Name	Description
`Use64BitHive` [page 83]	Gets or sets a flag indicating whether SapSetup operates on the 64-bit branch of the Windows registry.

NwEngine.Shell.CopyDirectory

Copies source directory to target directory with all content.

Syntax

```
$NwEngine.Shell.CopyDirectory([string]$sourceDir, [string]$target)
```

Arguments

`sourceDir` required. The path to the directory to copy.

`target` required. If the target is a directory, the target file name is the same as the source file name. If the target does not exist, or if it is a file, then this is the target file name.

Returns

Returns `[bool]$true` if the directory copy was successful, else it returns `[bool]$false`.

Remarks

Existing files will not be overwritten.

Sample Code

```
[string]$source = $NwEngine.Variables.ResolveString("%SapSrcDir%
\CustomerFiles\myfiles")
```

```
[string]$target = "C:\path\to\yourfiles"
if ($NwEngine.Shell.CopyDirectory($source, $target))
{
    $NwEngine.Context.Log.Write("Copying " + $source + " to " + $target + "
was successful.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Variables.ResolveString` [page 93]

Copies the directory `%SapSrcDir%\CustomerFiles\myfiles` to `C:\path\to\yourfiles` and writes a message to the log, if the directory copy succeeded.

NwEngine.Shell.CopyDirectoryEx

Copies source directory to target directory with all content.

Syntax

```
$NwEngine.Shell.CopyDirectoryEx([string]$sourceDir, [string]$target, [bool]
$overwrite)
```

Arguments

`sourceDir` required. The path to the directory to copy.

`target` required. If the target is a directory, the target file name is the same as the source file name. If the target does not exist, or if it is a file, then this is the target file name.

`overwrite` required. If set to `$true`, an existing file will be overwritten. If set to `$false`, existing files will not be overwritten.

Returns

Returns `[bool]$true` if the directory copy was successful, else it returns `[bool]$false`.

Remarks

Copies source directory to target directory with all content. When setting `[bool]$overwrite` to `[bool]$false`, this method behaves like `NwEngine.Shell.CopyDirectory(...)`. This method is useful to update files copied from `CustomerFiles` on the installation server.

Sample Code

Copies the directory `%SapSrcDir%\CustomerFiles\myfiles` to `C:\path\to\yourfiles`, overwriting existing files, and writes a message to the log, if the directory copy succeeded.

```
[string]$source = $NwEngine.Variables.ResolveString("%SapSrcDir%
\CustomerFiles\myfiles")
[string]$target = "C:\path\to\yourfiles"
if ($NwEngine.Shell.CopyDirectoryEx($source, $target, $true))
{
    $NwEngine.Context.Log.Write("Copying " + $source + " to " + $target + "
was successful.")
```

```
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Variables.ResolveString` [page 93]

NwEngine.Shell.CopyFile

Copies source file to target file on the file system.

Syntax

```
$NwEngine.Shell.CopyFile([string]$sourceFile, [string]$target)
```

Arguments

`sourceFile` required. The name of the file to copy.

`target` required. If the target is a directory, the target file name is the same as the source file name. If the target does not exist, or if it is a file, then this is the target file name.

Returns

Returns `[bool]$true` if file copy was successful, else it returns `[bool]$false`.

Remarks

Existing files will not be overwritten.

Sample Code

Copies the file `%SapSrcDir%\setup\netstart.exe` to directory `C:\path\to\setup` and writes a message to the log, if the file copy succeeded.

```
[string]$source = $NwEngine.Variables.ResolveString("%SapSrcDir%
\CustomerFiles\BrandingImage.png")
[string]$target = $NwEngine.Variables.ResolveString("%SapFrontEndDir%
\SapGui\BrandingImage.png")
if ($NwEngine.Shell.CopyFile($source, $target))
{
    $NwEngine.Context.Log.Write("Copying " + $source + " to " + $target + "
was successful.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Variables.ResolveString` [page 93]

NwEngine.Shell.CopyFileEx

Copies source file to target file on the file system.

Syntax

```
$NwEngine.Shell.CopyFileEx([string]$sourceFile, [string]$target, [bool]$overwrite)
```

Arguments

`sourceFile` required. The name of the file to copy.

`target` required. If the target is a directory, the target file name is the same as the source file name. If the target does not exist, or if it is a file, then this is the target file name.

`overwrite` required. If set to `$true`, an existing file will be overwritten. If set to `$false`, existing files will not be overwritten.

Returns

Returns `[bool]$true` if file copy was successful, else it returns `[bool]$true`.

Remarks

When setting `[bool]$overwrite` to `[bool]$false`, this method behaves like `NwEngine.Shell.CopyFile(...)` [page 60]. This method is useful to update files copied from `CustomerFiles` on the installation server.

Sample Code

Copies the file `%SapSrcDir%\CustomerFiles\BrandingImage.png` to `%SapFrontEndDir%\SapGui\BrandingImage.png` and writes a message to the log, if the file copy succeeded. If the file exists at the destination, it will be overwritten.

```
[string]$source = $NwEngine.Variables.ResolveString("%SapSrcDir%
\CustomerFiles\BrandingImage.png")
[string]$target = $NwEngine.Variables.ResolveString("%SapFrontEndDir%
\SapGui\BrandingImage.png")
if ($NwEngine.Shell.CopyFileEx($source, $target, $true))
{
    $NwEngine.Context.Log.Write("Copying " + $source + " to " + $target + "
was successful.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Variables.ResolveString` [page 93]

NwEngine.Shell.CreateDirectory

Creates a directory.

Syntax

`$NwEngine.Shell.CreateDirectory([string]$target)`

Arguments

`target` required. Path to the directory to create.

Returns

Returns `[bool]$true` if the creation was successful, else it returns `[bool]$false`.

Sample Code

Tries to create directory `C:\path\to\myDir` and writes the error message `"Creating directory C:\path\to\myDir failed."` to the log, if the creation failed.

```
[string]$target = "C:\path\to\myDir"
if (-not $NwEngine.Shell.CreateDirectory(target))
{
    $NwEngine.Context.Log.Write("Creating directory " + $target + " failed.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Variables.ResolveString` [page 93]

NwEngine.Shell.CreateTempDirectory

Creates a temporary directory.

Syntax

`$NwEngine.Shell.CreateTempDirectory()`

Returns

Returns the path of the created directory as `[string]`.

Remarks

The temporary directory is created in `$env:TEMP`. This is useful for more complex file operations that should not be done within the destination directory of the installation. For more information, see `NwEngine.Shell.DeleteDirectory` [page 63].

Sample Code

Creates a directory in `$env:TEMP` and finally deletes it again.

```
[string]$tempPath = $NwEngine.Shell.CreateTempDirectory()
# Do some file operations there
$NwEngine.Shell.DeleteDirectory($tempPath)
```

Reference

- `NwEngine.Shell.DeleteDirectory` [page 63]

NwEngine.Shell.DeleteDirectory

Deletes a directory.

Syntax

```
$NwEngine.Shell.DeleteDirectory([string]$target)
```

Arguments

`target` required. Path to the directory to delete.

Returns

Returns `[bool]$true` if the deletion was successful, else it returns `[bool]$false`.

Remarks

> ⚠ Caution
>
> All content of the directory will be deleted. Use with great care.

> {·} Sample Code
>
> Creates a directory in `$Env:TEMP` and returns the path to it. Afterwards the temporary directory is deleted. For more information, see `NwEngine.Shell.CreateTempDirectory` [page 62].

```
[string]$tempPath = $NwEngine.Shell.CreateTempDirectory()
# Do some file operations there
$NwEngine.Shell.DeleteDirectory($tempPath)
```

Reference

- `NwEngine.Shell.CreateTempDirectory` [page 62]

NwEngine.Shell.DeleteFile

Deletes a file.

Syntax

```
$NwEngine.Shell.DeleteFile([string]$target)
```

Arguments

`target` required. Path to the file to delete.

Returns

Returns `[bool]$true` if the deletion was successful, else it returns `[bool]$false`.

Sample Code

Deletes the file `C:\path\to\myFile.ext` and writes a message to the log, when the deletion was successful..

```
[string]$someFile = "C:\path\to\myFile.ext"
if ($NwEngine.Shell.DeleteFile(someFile))
{
    $NwEngine.Context.Log.Write("File " + $someFile + " deleted
successfully.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.DeleteRegKey

Note

On 64-bit Windows, SapSetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 83] property.

Deletes a registry key.

Syntax

`$NwEngine.Shell.DeleteRegKey([string]$key)`

Arguments

`key` required. Path to the registry key to delete.

Returns

Returns `[bool]$true` if the deletion was successful, else it returns `[bool]$false`.

Sample Code

Removes the killbit of `vsflex71.ocx` by deleting the corresponding registry key. Writes appropriate messages to the log file.

```
[string]$key = "Internet Explorer\ActiveX Compability\{C0A63B86-4B21-11D3-
BD95-D426EF2C7949}"
$NwEngine.Context.Log.Write("Removing the killbit for vsflex71.ocx.")
if ($NwEngine.Shell.RegKeyExist("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\" +
$key))
```

```
{
    if ($NwEngine.Shell.DeleteRegKey("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\"
+ $key))
    {
        $NwEngine.Context.Log.Write("Deleting the killbit for vsflex71.ocx
has succeeded.")
    }
    else
    {
        $NwEngine.Context.Log.Write("Deleting the killbit for vsflex71.ocx
has failed.")
    }
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Shell.DeleteRegKey` [page 64]

NwEngine.Shell.DeleteRegValue

> **Note**
>
> On 64-bit Windows, SapSetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 83] property.

> **Note**
>
> To access the `(Default)` value of a registry key, add a backslash `\` at the end of the registry key. To access the `(Default)` value from `HKLM\SOFTWARE\SAP`, add a backslash at the end: **`HKLM\SOFTWARE\SAP\`**.

Deletes a registry value.

Syntax

```
$NwEngine.Shell.DeleteRegValue([bool]$value, [bool]$valueName)
```

Arguments

`value` required. Path to the registry value to delete.

`valueName` optional. Name of the registry value to delete.

Returns

Returns `[bool]$true` if the deletion was successful, else it returns `[bool]$false`.

Sample Code

Removes the autostart of Java Update Scheduler by deleting the corresponding registry value. Writes appropriate messages to the log file.

```
[string]$value =
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\SunJavaUpdateShed"
$NwEngine.Context.Log.Write("Removing autostart of Java Update Scheduler")
if ($NwEngine.Shell.DeleteRegValue($value))
{
    $NwEngine.Context.Log.Write("Deleting the autostart of Java Update
Scheduler has succeeded.")
}
else
{
    $NwEngine.Context.Log.Write("Deleting the autostart of Java Update
Scheduler has failed.")
}
```

Sample Code

Optional parameter is required to delete a registry value which contains a backslash (‘\’). Otherwise `DeleteRegValue` tries to split registry key and value with the last backslash and will fail as the registry key cannot be found.

```
[string]$value =
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs"
[string]$valueName = "C:\Windows\System32\mfc140.dll"
$NwEngine.Context.Log.Write("Removing SharedDLL counter for " + $valueName)
if ($NwEngine.Shell.DeleteRegValue($value, $valueName))
{
    $NwEngine.Context.Log.Write("Deleting the registry value for " +
$valueName + " has succeeded.")
}
else
{
    $NwEngine.Context.Log.Write("Deleting the registry value for " +
$valueName + " has failed.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.DirectoryExist

Checks if a directory exists.

Syntax

```
$NwEngine.Shell.DirectoryExist([string]$target)
```

Arguments

`target` required. Path to the directory to check for existence.

Returns

Returns `[bool]$true` if the directory exists, else it returns `[bool]$false`.

Sample Code

Writes a line to the log if the file `C:\path\to\myDir` exists.

```
$someDir = "C:\path\to\myDir"
if ($NwEngine.Shell.DirectoryExist($someDir))
{
    $NwEngine.Context.Log.Write("File " + $someDir + " exists.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.EnumRegKeys

Note

On 64-bit Windows, SapSetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 83] property.

Returns a collection containing all keys below a specified registry path.

Syntax

`NwEngine.Shell.EnumRegKeys(String path)$NwEngine.Shell.EnumRegKeys([bool]$path)`

Arguments

`path` required. Path to the registry key to enumerate.

Returns

Returns a `collection` of the registry keys that are direct children of the specified path. Does not return the whole tree below the specified path recursively.

Sample Code

```
[string]$path =
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall"
foreach($key in $NwEngine.Shell.EnumRegKeys($path))
{
    $NwEngine.Context.Log.Write("Key: " + $key)
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.EnumRegValues

> **Note**
>
> On 64-bit Windows, SapSetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 83] property.

Returns a collection containing all values below a specified registry path.

Syntax

```
$NwEngine.Shell.EnumRegValues([string]$path)
```

Arguments

`path` required. Path to the registry key to enumerate.

Returns

Returns a `collection` of the registry values that are direct children of the specified path. Does not return the whole tree below the specified path recursively.

> **Sample Code**
>
> Enumerates all registry values in `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` on both the 32-bit hive and the 64-bit hive. Finally resets the registry mode back to 32-bit hive.

```
[string]$autorunPath = "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
foreach($autorunValue in $NwEngine.Shell.EnumRegValues($autorunPath))
{
    $NwEngine.Context.Log.Write("Autorun value: " + $autorunValue)
}
# Do not forget the 64-bit hive.
$NwEngine.Shell.Use64BitHive = $true
foreach($autorunValue in $NwEngine.Shell.EnumRegValues($autorunPath))
{
    $NwEngine.Context.Log.Write("Autorun value: " + $autorunValue)
}
# Switch back to 32-bit hive.
$NwEngine.Shell.Use64BitHive = $false
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `Use64BitHive` [page 83]

NwEngine.Shell.Execute

Executes a command line.

Syntax

```
$NwEngine.Shell.Execute([string]$cmdLine, [bool]$async)
```

Arguments

`cmdline` required. Command line to execute.

`async` required. If `async` is set to `[bool]$true`, the process will be started asynchronously. If `async` is set to `[bool]$false`, SapSetup waits for the process to finish.

Returns

Returns `[bool]$true` if the process has finished, else `[bool]$false`.

Sample Code

Executes `notepad.exe` with file `C:\path\to\readme.txt`, if SapSetup is running in interactive mode. `notepad.exe` is executed asynchronously, the file will be shown and SapSetup will finish the installation.

```
[string]$cmdLine = "notepad.exe C:\Path\to\readme.txt"
if (-not $NwEngine.Context.Silent)
{
    $NwEngine.Context.Log.Write("Showing readme file.")
    $NwEngine.Shell.Execute($cmdLine, $true)
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.ExecuteEx

Executes a command line, optionally without displaying a window.

Syntax

```
$NwEngine.Shell.ExecuteEx([string]$cmdLine, [bool]$async, [bool]$visible)
```

Arguments

`cmdline` required. Command line to execute.

`async` required. If `async` is set to `[bool]$true`, the process will be started asynchronously. If `async` is set to `[bool]$false`, SapSetup waits for the process to finish.

`visible` required. If `visible` is set to `[bool]$false`, no window will be shown. This is especially useful for executing batch processes.

Returns

Returns `[bool]$true` if the process has finished, else `[bool]$false`.

Sample Code

Executes `%SapSrcDir%\CustomerFiles\mybatch.cmd` without showing the command prompt window. Waits for the execution before continuing.

```
[string]cmdLine = "%SapSrcDir%\CustomerFiles\mybatch.cmd"
$NwEngine.Context.Log.Write("Executing mybatch.cmd...")
[bool]retVal = $NwEngine-Shell.ExecuteEx($cmdLine, $false, $true)
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.ExecuteRc

Executes a command line, optionally without displaying a window.

Syntax

```
$NwEngine.Shell.ExecuteRC([string]$cmdLine, [bool]$visible)
```

Arguments

`cmdline` required. Command line to execute.

`visible` required. If `visible` is set to `[bool]$false`, no window will be shown. This is especially useful for executing batch processes.

Returns

Returns the return code of the executed command as `[string]`.

Sample Code

Executes `%SapSrcDir%\CustomerFiles\mybatch.cmd` without showing the command prompt window. Waits for the execution before continuing, and prints the return code of the executed command to the log file.

```
[string]cmdLine = "%SapSrcDir%\CustomerFiles\mybatch.cmd"
$NwEngine.Context.Log.Write("Executing mybatch.cmd...")
[string]returnCode = $NwEngine.Shell.ExecuteRC($cmdLine, $true)
$NwEngine.Context.Log.Write("Executing mybatch.cmd finished with " +
$returnCode + ".")
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.FileExist

Checks if a file exists.

Syntax

`$NwEngine.Shell.FileExist([string]$target)`

Arguments

`target` required. Path to the file to check for existence.

Returns

Returns `[bool]$true` if the file exists, else it returns `[bool]$false`.

Sample Code

Writes a line to the log if the file `C:\path\to\myFile.ext` exists.

```
[string]$someFile = "C:\path\to\myFile.ext"
if ($NwEngine.Shell.FileExist($someFile))
{
    $NwEngine.Context.Log.Write("File " + $someFile + " exists.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.GetDriveType

Returns the type of a drive.

Syntax

`$NwEngine.Shell.GetDriveType([string]$drive)`

Arguments

`drive` required. Path to the drive to check.

Returns

Returns a `[int]` indicating the drive type:

- 0: `DriveTypeFixed` (e.g. HDD)
- 1: `DriveTypeRemovable` (e.g. floppy)
- 2: `DriveTypeRemote` (e.g. network drive)
- 3: `DriveTypeCDROM` (e.g. DVD-ROM)
- 64: `DriveTypeUnknown`

Sample Code

Writes the numeric type of drive `C:` to the log.

```
$NwEngine.Context.Log.Write("Type of drive C: is: " +
$NwEngine.Shell.GetDriveType("C:"))
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.GetFileExt

Returns the extension of a file name.

Syntax

```
$NwEngine.Shell.GetFileExt([string]$file)
```

Arguments

`file` required. File to get the extension of.

Returns

Returns the file extension of a file as `[string]`.

Sample Code

Writes the file extension of file `someFile` to the log.

```
[string]$someFile = "C:\path\to\myFile.ext"
$NwEngine.Context.Log.Write("Extension of file " + $someFile + " is " +
$NwEngine.Shell.GetFileExt($someFile))
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.GetFileName

Returns the file extension of a file.

Syntax

```
$NwEngine.Shell.GetFileName([string]$path)
```

Arguments

`path` required. Path to get the file name of.

Returns

Returns the file name of a path as `[string]`.

Sample Code

Writes the file name of the SapSetup executable to the log.

```
[string]$sapsetup = $NwEngine.Variables.ResolveString("%SapSrcDir%
\Setup\NwSapSetup.exe")
$NwEngine.Context.Log.Write("Filename of SapSetup is " +
$NwEngine.Shell.GetFileName($sapsetup))
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Variables.ResolveString` [page 93]

NwEngine.Shell.GetFileSize

Returns the size of a file in bytes.

Syntax

```
$NwEngine.Shell.GetFileSize([string]$file)
```

Arguments

`file` required. File to get the size of.

Returns

Returns the size of a file as `[int]` in bytes.

Sample Code

Checks the size of file `%SapSrcDir%\Setup\NwSapSetup.exe` and writes an info to the log, if the file has size 0.

```
[string]$myFile = $NwEngine.Variables.ResolveString("%SapSrcDir%
\Setup\NwSapSetup.exe")
[int]$size = $NwEngine.Shell.GetFileSize($myFile)
if ($size -eq 0)
{
    $NwEngine.Context.Log.Write($myFile + " is empty.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Variables.ResolveString` [page 93]

NwEngine.Shell.GetFileVersion

Returns the version of a file.

Syntax

```
$NwEngine.Shell.GetFileVersion([string]$file)
```

Arguments

`file` required. File to get the version of.

Returns

Returns the version of a file as `[string]`.

Sample Code

Resolves the path to the 64-bit `cmd.exe`. Writes the file version of file `cmd.exe` to the log.

```
[string]$cmdExe = $NwEngine.Variables.ResolveString("%WinSysDir64%\cmd.exe")
$NwEngine.Context.Log.Write("Version of cmd.exe is " +
$NwEngine.Shell.GetFileVersion($cmdExe))
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Variables.ResolveString` [page 93]

NwEngine.Shell.GetPath

Returns the path to a file.

Syntax

```
$NwEngine.Shell.GetPath([string]$file)
```

Arguments

`file` required. File to get the path of.

Returns

Returns the path to a file as `[string]`.

Sample Code

Resolves the path to the 64-bit `cmd.exe`. Writes the path to file `cmd.exe` to the log.

```
[string]$cmdExe = $NwEngine.Variables.ResolveString("%WinSysDir64%\cmd.exe")
$NwEngine.Context.Log.Write("Path to cmd.exe is " +
$NwEngine.Shell.GetPath($cmdExe))
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Variables.ResolveString` [page 93]

NwEngine.Shell.GetRegValue

> **Note**
>
> On 64-bit Windows, SapSetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 83] property.

> **Note**
>
> To access the `(Default)` value of a registry key, just add a backslash `\` at the end of the registry key. To access the `(Default)` value from `HKLM\SOFTWARE\SAP`, add a backslash at the end: `HKLM\SOFTWARE\SAP\`.

Returns the value of a registry key.

Syntax

```
$NwEngine.Shell.GetRegValue([string]$key, [string]$valueName)
```

Arguments

`key` required. Registry key to get the value of.

`valueName` optional. Name of the registry value to get.

Returns

Returns the value of a registry key as `[string]`.

> **Sample Code**
>
> If the key `HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon` exists, its value is written to the log.

```
[string]$regVal = "HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon"
if ($NwEngine.Shell.RegValueExist($regVal))
{
    [string]$regValue = $NwEngine.Shell.GetRegValue($regVal)
    $NwEngine.Context.Log.Write("Registry value of " + $regVal + " is: " +
$regValue)
}
```

Sample Code

Optional parameter is required to retrieve a registry value which contains a backslash (\). Otherwise `GetRegValue` tries to split registry key and value with the last backslash and will fail as the registry key cannot be found.

```
[string]$regKey = "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs"
[string]$regValueName = "C:\WINDOWS\System32\mfc140.dll"
if ($NwEngine.Shell.RegValueExist($regKey, $regValueName))
{
    [string]$value = $NwEngine.Shell.GetRegValue($regKey, $regValueName)
    $NwEngine.Context.Log.Write("Registry value of " + $regKey + "\" +
$regValueName + " is: " + $value)
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Shell.RegValueExist` [page 79]

NwEngine.Shell.GetShortPath

Returns the short path in 8.3 notation of a path.

Syntax

```
$NwEngine.Shell.GetShortPath([string]$path)
```

Arguments

`path` required. Path to get the short path of.

Returns

Returns the short path in 8.3 notation of a path as `[string]`.

Sample Code

Resolves the path to the directory `CustomerFiles` on the installation server. Writes the path to it to the log.

```
[string]$customerFiles = $NwEngine.Variables.ResolveString("%SapSrcDir%
\CustomerFiles")
$NwEngine.Context.Log.Write("Short path to " + $customerFiles + " is " +
$NwEngine.Shell.GetShortPath($customerFiles) + ".")
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Variables.ResolveString` [page 93]

NwEngine.Shell.GetSystemUtcTime

Returns the system UTC time.

Syntax

```
$NwEngine.Shell.GetSystemUtcTime()
```

Returns

Returns the system UTC time as `[string]`.

Sample Code

Writes the current system UTC time to the log.

```
$NwEngine.Context.Log.Write("Current UTC time is " +
$NwEngine.Shell.GetSystemUtcTime())
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.IsExeRunning

Checks if a specific executable is running.

Syntax

```
$NwEngine.Shell.IsExeRunning([string]$filename)
```

Arguments

`filename` required. File that is checked if it is running as executable.

Returns

Returns `[bool]$true` if the file is running as executable.

Sample Code

Writes a line to the log if `saplogon.exe` is running.

```
if ($NwEngine.Shell.IsExeRunning("saplogon.exe"))
{
    $NwEngine.Context.Log.Write("SAPlogon is still running.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.IsPathEqual

Compares two paths for equality.

Syntax

```
$NwEngine.Shell.IsPathEqual([string]$path1, [string]$path2)
```

Arguments

`path1` required. First path to compare.

`path2` required. Second path to compare.

Returns

Returns `[bool]$true` if the two paths point to the same location, otherwise returns `[bool]$false`.

Remarks

This method is helpful to compare the following:

- Paths to the network
- Paths that contain environment variables
- Absolute and relative paths

Sample Code

Retrieve the value `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\NwSapSetup.exe\(Default)` from the Windows registry and compares it to the path `%SapSrcDir%\Setup\NwSapSetup.exe`. If the two paths point to the same location, write an entry to the log.

```
[string]$pathFromRegistry =
$NwEngine.Shell.GetRegValue("HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Ap
p Paths\NwSapSetup.exe\")
[string]$pathOnDisk = $NwEngine.Variables.ResolveString("%SapSrcDir%
\Setup\NwSapSetup.exe")
if ($NwEngine.Shell.IsPathEqual($pathFromRegistry, $pathOnDisk))
{
    $NwEngine.Context.Log.Write("Path in registry is same as " + $pathOnDisk)
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Shell.GetRegValue` [page 75]
- `$NwEngine.Variables.ResolveString` [page 93]

NwEngine.Shell.RegKeyExist

Note

On 64-bit Windows, SapSetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written

to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 83] property.

Checks if a registry key exists.

Syntax

```
$NwEngine.Shell.RegKeyExist([string]$key)
```

Arguments

`key` required. Path to the registry key to check.

Returns

Returns `[bool]$true` if the registry key exists, else it returns `[bool]$false`.

Sample Code

Writes a line to the log if registry key `HKLM\SOFTWARE\SAP` exists.

```
if ($NwEngine.Shell.RegKeyExist("HKLM\SOFTWARE\SAP"))
{
    $NwEngine.Context.Log.Write("Registry key HKLM\SOFTWARE\SAP exists.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.RegValueExist

Note

On 64-bit Windows, SapSetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 83] property.

Note

To access the `(Default)` value of a registry key, just add a backslash `\` at the end of the registry key. To access the `(Default)` value from `HKLM\SOFTWARE\SAP`, add a backslash at the end: `HKLM\SOFTWARE\SAP\`.

Checks if a registry value exists.

Syntax

```
$NwEngine.Shell.RegValueExist([string]$value, [string]$valueName)
```

Arguments

`value` required. Path to the registry value to check.

`valueName` optional. Name of the registry value to check.

Returns

Returns `[bool]$true` if the registry value exists, else it returns `[bool]$false`.

Sample Code

If the key `HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon` exists, write its value to the log.

```
[string]$regVal = "HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon"
if ($NwEngine.Shell.RegValueExist($regVal))
{
    [string]$regValue = $NwEngine.Shell.GetRegValue($regVal)
    $NwEngine.Context.Log.Write("Registry value of " + $regVal + " is: " +
$regValue)
}
```

Sample Code

Optional parameter is required to check the existence of a registry value which contains a backslash (`\`) . Otherwise `GetRegValue` tries to split registry key and value with the last backslash and will fail as the registry key cannot be found.

```
[string]$regKey = "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs"
[string]$regValueName = "C:\WINDOWS\System32\mfc140.dll"
if ($NwEngine.Shell.RegValueExist($regKey, $regValueName))
{
    [string]$value = $NwEngine.Shell.GetRegValue($regKey, $regValueName)
    $NwEngine.Context.Log.Write("Registry value of " + $regKey + "\" +
$regValueName + " is: " + $value)
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Shell.GetRegValue` [page 75]

NwEngine.Shell.SetRegValue

Note

On 64-bit Windows, SapSetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 83] property.

> ### Note
>
> To access the `(Default)` value of a registry key, just add a backslash `\` at the end of the registry key. To access the `(Default)` value from `HKLM\SOFTWARE\SAP`, add a backslash at the end: `HKLM\SOFTWARE\SAP\`.

Sets a registry value.

Syntax

```
$NwEngine.Shell.SetRegValue([string]$valuePath, [string]$valueType, [string]$value, [string]$valueName)
```

Arguments

`valuePath` required. Path to the registry value to set.

`valueType` required. Type of the registry value to set. Must be one of the following:

- `REG_SZ`
- `REG_EXPAND_SZ`
- `REG_MULTI_SZ`
- `REG_BINARY`
- `REG_DWORD`

`value` required. Value to set.

`valueName` optional. Name of the registry value to set.

Returns

Returns `[bool]$true` if the registry value was set successfully, else it returns `[bool]$false`.

> ### Sample Code
>
> Sets the registry value `HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon` to type `REG_DWORD` and value `1`.
>
> ```
> $NwEngine.Shell.SetRegValue("HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon", "REG_DWORD", "1")
> ```

> ### Sample Code
>
> Optional parameter is required to set a registry value which contains a backslash (`\`). Otherwise `SetRegValue` tries to split registry key and value with the last backslash and will fail as the registry key cannot be found.
>
> ```
> if ($NwEngine.Shell.SetRegValue("HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs", "REG_DWORD", "1", "C:\Windows\System32\mfc140.dll"))
> {
>     $NwEngine.Context.Log.Write("Registry value successfully created.")
> }
> ```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Shell.Sleep

Makes SapSetup sleep for a specified time.

Syntax

```
$NwEngine.Shell.Sleep([int]$milliseconds)
```

Arguments

`milliseconds` required. Number of milliseconds SapSetup shall sleep.

Remarks

This is useful when the script does something that has side effects, like shutting down a Windows service, which needs a few seconds, and there is no possibility to wait for it.

{·} Sample Code

Makes SapSetup sleep for 5 seconds.

```
$NwEngine.Shell.Sleep(5000)
```

NwEngine.Shell.TerminateProcessInstances

Terminates all running processes with a specific executable.

Syntax

```
$NwEngine.Shell.TerminateProcessInstances([string]$filename)
```

Arguments

`filename` required. Name of the executable to terminate.

Remarks

⚠ Caution

This call terminates all processes that run with the specified executable. Use it only if there is definitely no other way to shut down a running process in a controlled way. Use with great care.

{·} Sample Code

Writes a line to the log if `saplogon.exe` is running.

```
if ($NwEngine.Shell.IsExeRunning("saplogon.exe"))
```

```
{
    $NwEngine.Context.Log.Write("SAPlogon is still running. Killing all
instances of it.")
    $NwEngine.Shell.TerminateProcessInstances("saplogon.exe")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]
- `$NwEngine.Shell.IsExeRunning` [page 77]

NwEngine.Shell.Use64BitHive

Gets or sets a flag indicating whether SapSetup operates on the 64-bit branch of the Windows registry.

Syntax

`[bool]$NwEngine.Shell.Use64BitHive`

Remarks

SapSetup is a 32-bit application. By default all operations on the Windows registry are done on the 32-bit hive. Use this property to access the 64-bit hive. When `Use64BitHive` is set to `[bool]$true`, following registry operations will be done on the 64-bit branch of the registry. When `Use64BitHive` is set to `[bool]$false`false, following registry operations will be done on the 32-bit branch of the registry.

Sample Code

```
$NwEngine.Shell.Use64BitHive = $true
# Here be operations on the 64-bit hive of the registry.
# Always reset to 32-bit give!
$NwEngine.Shell.Use64BitHive = $false
```

8.1.2 NwEngine.Context

`$NwEngine.Context` provides access to properties to determine operating system, computer name, rights elevation status, command line arguments and more.

Methods

Method Name	Description
`Elevated` [page 84]	Gets a flag indicating whether this installer instance is executed with elevated rights from LSH.
`LCID` [page 85]	Gets the system locale identifier of the current workstation.

Method Name	Description
Reboot [page 85]	Gets or sets a flag indicating whether the reboot flag should be set.
Silent [page 85]	Gets or sets a flag indicating whether the installation mode was started in silent mode.
WinVista [page 86]	Gets or sets a flag indicating whether running on a Windows Vista or Windows Server 2008 machine.
Windows7 [page 86]	Gets or sets a flag indicating whether running on a Windows 7 or Windows Server 2008 R2 machine.
Windows8 [page 87]	Gets or sets a flag indicating whether running on a Windows 8 or Windows Server 2012 machine.
Windows81 [page 87]	Gets or sets a flag indicating whether running on a Windows 8.1 or Windows Server 2012 R2 machine.
Windows10 [page 88]	Gets or sets a flag indicating whether running on a Windows 10 or Windows Server 2016, 2019 or 2022 machine.
Windows11 [page 88]	Gets or sets a flag indicating whether running on a Windows 11 or Windows Server 2025 machine.
WkstaName [page 89]	Gets the NetBIOS name of the current workstation.
WTSClient [page 89]	Gets a flag indicating whether running on a WTS Client.
WTSServer [page 89]	Gets a flag indicating whether running on a WTS Server.

NwEngine.Context.Elevated

Gets a flag indicating whether this installer instance is executed with elevated rights from LSH.

Syntax

```
[bool]$NwEngine.Context.Elevated
```

Sample Code

Writes an entry to the log, if running elevated.

```
if ($NwEngine.Context.Elevated)
{
    $NwEngine.Context.Log.Write("We are running with elevation.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.LCID

Gets the hosting workstation's system locale identifier.

Syntax

```
[string]$NwEngine.Context.LCID
```

Sample Code

Writes the system locale identifier to the log.

```
$NwEngine.Context.Log.Write("System locale identifier is " +
$NwEngine.Context.LCID)
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.Reboot

Gets or sets a flag indicating whether the engine's reboot flag should be set.

Syntax

```
[bool]$NwEngine.Context.Reboot
```

Sample Code

The procedure will require a reboot after it has finished. Writes an according entry to the log.

```
$NwEngine.Context.Reboot = $true
$NwEngine.Context.Log.Write("Reboot will be required."
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.Silent

Gets or sets a value indicating whether this installation mode was started in silent mode.

Syntax

```
[bool]$NwEngine.Context.Silent
```

Sample Code

Writes an entry to the log, if running in silent mode.

```
if ($NwEngine.Context.Silent)
{
    $NwEngine.Context.Log.Write("We are running in silent mode, showing no
dialogs.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.WinVista

Gets a flag indicating whether running on a Windows Vista or Windows Server 2008 machine.

Syntax

```
[bool]$NwEngine.Context.WinVista
```

Sample Code

Writes an entry to the log, if running on Windows Vista or Windows Server 2008.

```
if ($NwEngine.Context.WinVista)
{
    $NwEngine.Context.Log.Write("We are running on Windows Vista or Windows
Server 2008.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.Windows7

Gets a flag indicating whether running on a Windows 7 or Windows Server 2008 R2 machine.

Syntax

```
[bool]$NwEngine.Context.Windows7
```

Sample Code

Writes an entry to the log, if running on Windows 7 or Windows Server 2008 R2.

```
if ($NwEngine.Context.Windows7)
{
    $NwEngine.Context.Log.Write("We are running on Windows Vista or Windows
Server 2008 R2.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.Windows8

Gets a flag indicating whether running on a Windows 8 or Windows Server 2012 machine.

Syntax

```
[bool]$NwEngine.Context.Windows8
```

Sample Code

Writes an entry to the log, if running on Windows 8 or Windows Server 2012.

```
if ($NwEngine.Context.Windows8)
{
    $NwEngine.Context.Log.Write("We are running on Windows 8 or Windows
Server 2012.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.Windows81

Gets a flag indicating whether running on a Windows 8.1 or Windows Server 2012 R2 machine.

Syntax

```
[bool]$NwEngine.Context.Windows81
```

Sample Code

Writes an entry to the log, if running on Windows 8.1 or Windows Server 2012 R2.

```
if ($NwEngine.Context.Windows81)
{
    $NwEngine.Context.Log.Write("We are running on Windows 8.1 or Windows
Server 2012 R2.")
```

```
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.Windows10

Gets a flag indicating whether running on a Windows 10 or Windows Server 2016, 2019 or 2022 machine.

Syntax

```
[bool]$NwEngine.Context.Windows10
```

Sample Code

Writes an entry to the log, if running on Windows 10 or Windows Server 2016, 2019 or 2022.

```
if ($NwEngine.Context.Windows10)
{
    $NwEngine.Context.Log.Write("We are running on Windows 10 or Windows
Server 2016, 2019 or 2022.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.Windows11

Gets a flag indicating whether running on a Windows 11 or Windows Server 2025 machine.

Syntax

```
[bool]$NwEngine.Context.Windows11
```

Sample Code

Writes an entry to the log, if running on Windows 11 or Windows Server 2025.

```
if ($NwEngine.Context.Windows11)
{
    $NwEngine.Context.Log.Write("We are running on Windows 11 or Windows
Server 2025.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.WkstaName

Gets the NetBIOS name of the current workstation.

Syntax

```
[string]$NwEngine.Context.WkstaName
```

Sample Code

Writes `Name of this workstation: <workstationName>` to the log file.

```
[string]$workstationName = $NwEngine.Context.WkstaName
$NwEngine.Context.Log.Write("Name of this workstation: " + $workstationName)
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.WTSClient

Gets a flag indicating whether this installer instance is executed on a WTS Client.

Syntax

```
[bool]$NwEngine.Context.WTSClient
```

Sample Code

Writes an entry to the log, if running on Windows Terminal Service Client.

```
if ($NwEngine.Context.WTSClient)
{
    $NwEngine.Context.Log.Write("We are running on Windows Terminal Service
client.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Context.WTSServer

Gets a flag indicating whether this installer instance is executed on a WTS Server.

Syntax

```
[bool]$NwEngine.Context.WTSServer
```

Sample Code

Writes an entry to the log, if running on Windows Terminal Service Server.

```
if ($NwEngine.Context.WTSServer)
{
    $NwEngine.Context.Log.Write("We are running on Windows Terminal Service
server.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

8.1.3 NwEngine.Context.Log

`$NwEngine.Context.Log` writes information texts, warnings, or errors to the log.

Procedure Name	Description
`Write` [page 90]	Writes a message into the current log file
`WriteError` [page 91]	Writes a message marked as error into the log file. Entry is marked as `"E"`. SapSetup return code will be not 0. Installation procedure will be handled as not successfully finished.
`WriteWarning` [page 91]	Writes a message marked as warning into the log file. Entry is marked as `"W"`.

NwEngine.Context.Log.Write

Writes a message to the log file.

Syntax

```
$NwEngine.Context.Log.Write([string]$message)
```

Arguments

`message` required. The message to write to the log file.

Returns

This method does not return anything.

Sample Code

Writes `"Hello World!"` to the log file.

```
$NwEngine.Context.Log.Write("Hello World!")
```

NwEngine.Context.Log.WriteError

Writes a message to the log file. Entry is marked as `"E"`.

Syntax

```
$NwEngine.Context.Log.WriteError([string]$errorMessage)
```

Arguments

`errorMessage` required. The error message to write to the log file.

Returns

This method does not return anything.

Remarks

Writes a message into the current log file as error. Installation procedure will be handled as not successfully finished, and SapSetup will exit with return code other than 0. This is useful to check prerequisites and make the installation process fail if the prerequisites are not fulfilled, so that it can be run again

Sample Code

Writes `"We will fail!"` to the log file, marked as error with `"E"`.

```
if ($somethingIsWrong)
{
    $NwEngine.Context.Log.WriteError("We will fail!")
}
```

NwEngine.Context.Log.WriteWarning

Writes a warning message to the log file.

Syntax

```
$NwEngine.Context.Log.WriteWarning([string]$warningMessage)
```

Arguments

`warningMessage` required. The warning message to write to the log file.

Returns

This method does not return anything.

Remarks

Writes a message into the current log file as warning. This does not change the return code of SapSetup. This is useful if you want to note important things down during the installation, and later analyze the log file of SapSetup. End users will never see this message.

Sample Code

Writes `"We might fail!"` to the log file, marked as warning with `"W"`.

```
if ($somethingIsStrange)
{
    $NwEngine.Context.Log.WriteWarning("We might fail!")
}
```

8.1.4 NwEngine.Variables

`$NwEngine.Variables` comprises methods to access variable values and resolve strings that contain variables.

Methods for Variable Resolution

Method Name	Description
`GetValue` [page 93]	Returns the value of the variable.
`ResolveString` [page 93]	Resolves all variables in given parameter and returns the resolved string.

Remarks

The variable `<%SapSrcDir%>` points to the root directory of the current installation source. Regardless if it is an installation server or a single-file installer of a package, below `<%SapSrcDir%>` you find the same directory structure. The directory `%SapSrcDir%\CustomerFiles` is always included in every package. It is useful if you want to distribute your custom files with packages.

The package configuration in `NwSapSetupAdmin` shows additional variables that are available in the package. You can set the values of variables for packages there, and you can access these variables in package event scripts. SapSetup provides a comprehensive list of additional variables containing information, among others, about the following:

- System directories
- Operating system
- Installed .NET runtimes
- Installed Java runtimes

- Installed Microsoft products
- Installed Adobe products

For the full list of all variables, start SapSetup and cancel it on the first dialog. Check the log file, look for the line `Log Variables Collection`. Do this on a system where no SapSetup products are installed, as installing products persists some variables on the computer.

NwEngine.Variables.GetValue

Returns the value of a variable.

Syntax

```
$NwEngine.Variables.GetValue([string]$variableName)
```

Arguments

`variableName` required. Name of the variable to get the value of.

Returns

Returns the value of the variable as `[string]`.

Sample Code

Gets the value of variable `<Wow64Mode>` and writes it to the log.

```
$winmode = $NwEngine.Variables.GetValue("Wow64Mode")
if ($winmode -eq "true")
{
    $NwEngine.Context.Log.Write("Value of variable Wow64Mode is " + $winmode
+ ". That means this Windows is 64 bit.")
}
else
{
    $NwEngine.Context.Log.Write("Value of variable Wow64Mode is " + $winmode
+ ". That means this Windows is 32 bit.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

NwEngine.Variables.ResolveString

Resolves all variables contained in given parameter and returns the resolved string.

Syntax

```
$NwEngine.Variables.ResolveString([string]$stringToResolve)
```

Arguments

`stringToResolve` required. A text that may contain SapSetup variables like `<%variableName%>`.

Returns

Returns the provided string with all SapSetup variables replaced by their values.

Sample Code

Resolves the variable `<WinSysDir>` in the string and writes the resolved string to the log file.

```
[string]$toResolve = "%WinSysDir%\drivers\etc\hosts"
[string]$resolved = $NwEngine.Variables.ResolveString($toResolve)
$NwEngine.Context.Log.Write("Resolved string is " + $resolved)
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

8.1.5 NwSapSetupATLCommon.TextFileParser

`$NwSapSetupATLCommon.TextFileParser` offers a method for reading and writing text files and searching and modifying text in text files.

Methods

Method Name	Description
`FileName` [page 95]	Returns the name of the parsed text file.
`Parse` [page 95]	Reads a text file from disk. Returns `[bool]$true` if file exists and can be read or `[bool]$false` if the file does not exist.
`ReplaceLine` [page 96]	Replaces all occurrences of the lines that match `LineToReplace` with the replacement string.
`ReplaceLineEx` [page 97]	Replaces all lines that contain the given string by the given replacement.
`DoesLineExist` [page 98]	Returns `[bool]$true` if one or more lines in the file completely match the given value, else `[bool]$false`.
`DoesStringExist` [page 98]	Returns `[bool]$true` if one or more lines in the file contain the given value, else `[bool]$false`.
`Save` [page 99]	Saves the file with the given file name. Returns `[bool]$true` if the file was saved successfully, else `[bool]$false`.

Method Name	Description
`AppendLine` [page 100]	Appends the given string at the end of the text file.
`NumLines` [page 100]	Returns the number of lines in the file.
`Line` [page 101]	Returns the line with the given line index.
`RemoveLine` [page 102]	Removes all lines from the text file that match the given line.

<$objTextFileParser>.FileName

Syntax

Returns the name of the opened file.

```
<$objTextFileParser>.FileName()
```

Returns

Returns the file name as `[string]`.

Remarks

If no file was parsed `FileName()` returns `$null`.

Sample Code

Returns the file name of a text file parser.

```
$objTextFileParser = New-Object -ComObject
"NwSapSetupATLCommon.TextFileParser"
$objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services")
$NwEngine.Context.Log.Write($objTextFileParser.FileName)
```

Reference

- `<$objTextFileParser>.Parse` [page 95]
- `$NwEngine.Context.Log.Write` [page 90]

<$objTextFileParser>.Parse

Reads a text file from disk.

Syntax

```
<$objTextFileParser>.Parse([string]$filename)
```

Arguments

`filename` required. Path to the text file to read.

Returns

Returns `[bool]$true` if file could be successfully read, `[bool]$false` if not.

Sample Code

Parse a file from disk and write appropriate message to the log file.

```
$objTextFileParser = New-Object -ComObject
"NwSapSetupATLCommon.TextFileParser"
if ($objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services"))
{
    $NwEngine.Context.Log.Write("File " + $objTextFileParser.FileName + "
successfully parsed.")
}
else
{
    $NwEngine.Context.Log.Write("Could not read file " +
$objTextFileParser.FileName + ".")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

<$objTextFileParser>.ReplaceLine

Replaces all occurrences of the lines that match `LineToReplace` with the replacement string.

Syntax

```
<$objTextFileParser>.ReplaceLine([string]$oldLine, [string]$newLine)
```

Arguments

`oldLine` required. Lines that match this text will be replaced.

`newLine` required. New content of the matching lines.

Returns

This method does not return anything.

Remarks

The whole line is compared to the parameter `oldLine`. Use `ReplaceLineEx` [page 97] if you want to match parts of the lines.

If more than one line matches the given value, all lines are replaced.

Sample Code

Replaces line by its content.

```
$objTextFileParser = New-Object -ComObject
"NwSapSetupATLCommon.TextFileParser")
if ($objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services"))
{
    $objTextFileParser.ReplaceLine("sappcd 3500/tcp", "sappcd 3500/udp #this
line is changed")
    $objTextFileParser.Save("C:\Windows\System32\drivers\etc\services")
}
```

Reference

- `<$objTextFileParser>.Parse` [page 95]
- `<$objTextFileParser>.Save` [page 99]
- `$NwEngine.Context.Log.Write` [page 90]

<$objTextFileParser>.ReplaceLineEx

Replaces all occurrences of the lines where a part of the content matches the search criteria with the replacement string.

Syntax

```
<$objTextFileParser>.ReplaceLineEx([string]$searchString, [string]$newLine)
```

Arguments

`searchString` required. Search criteria that contain this text will be replaced.

`newLine` required. New content of the replaced lines.

Returns

This method does not return anything.

Remarks

All lines that contain the search criteria will be replaced by the given value.

Sample Code

Replaces line by part of its content.

```
$objTextFileParser = New-Object -ComObject
"NwSapSetupATLCommon.TextFileParser"
if ($objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services"))
{
    $objTextFileParser.ReplaceLineEx("sppcd 3500", "sappcd 3500/udp #this
line is changed")
    $objTextFileParser.Save("C:\Windows\System32\drivers\etc\services")
}
```

Reference

- `<$objTextFileParser>.Parse` [page 95]
- `<$objTextFileParser>.Save` [page 99]
- `$NwEngine.Context.Log.Write` [page 90]

<$objTextFileParser>.DoesLineExist

Checks the existence of a line.

Syntax

```
<$objTextFileParser>.DoesLineExist([string]$searchCriteria)
```

Arguments

`searchCriteria` required. Content of line to be verified.

Returns

Returns `[bool]$true` if line could be found, else `[bool]$false`.

Sample Code

Checks existence of a line by comparing complete content of line.

```
$objTextFileParser = New-Object -ComObject
"NwSapSetupATLCommon.TextFileParser"
if ($objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services"))
{
    if ($objTextFileParser.DoesLineExist("login 513/tcp #Remote Login"))
    {
        $NwEngine.Context.Log.Write("Line for Remote login service found.")
    }
}
```

Reference

- `<$objTextFileParser>.Parse` [page 95]
- `$NwEngine.Context.Log.Write` [page 90]

<$objTextFileParser>.DoesStringExist

Checks whether the given string exists in the file.

Syntax

```
<$objTextFileParser>.DoesStringExist([string]$searchCriteria)
```

Arguments

`searchCriteria` required. String to be searched for.

Returns

Returns `[bool]$true` if string could be found, else `[bool]$false`.

Sample Code

Checks the existence of a line by a search string.

```
$objTextFileParser = New-Object -ComObject
"NwSapSetupATLCommon.TextFileParser"
if ($objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services"))
{
    if ($objTextFileParser.DoesStringExist("513/tcp"))
    {
        $NwEngine.Context.Log.Write("Line for Remote login service found.")
    }
}
```

Reference

- `<$objTextFileParser>.Parse` [page 95]
- `$NwEngine.Context.Log.Write` [page 90]

<$objTextFileParser>.Save

Saves the data of the `TextFileParser` object to a file.

Syntax

```
<$objTextFileParser>.Save([string]$filename)
```

Arguments

`filename` required: Name of the file where the `TextFileParser` stores its data.

Returns

Returns `[bool]$true` if successfully saved, else `[bool]$false`.

Remarks

If the file to be saved already exists, the existing file will be deleted before the new content is saved.

Sample Code

Saves buffer of `TextFileParser` object to disk.

```
if ($objTextFileParser.Save("C:\Windows\System32\drivers\etc\services"))
{
    $NwEngine.Context.Log.Write("Services file successfully saved.")
}
```

Reference

- `$NwEngine.Context.Log.Write` [page 90]

<$objTextFileParser>.AppendLine

Appends a line to the text buffer of the `TextFileParser` object.

Syntax

```
<$objTextFileParser>.AppendLine([string]$line)
```

Arguments

`line` required: Text to be appended to the buffer.

Returns

This method does not return anything.

Remarks

`AppendLine` appends the text to the internal buffer of the `TextFileParser`. If you want to append an existing file, you have to parse it in the first step.

Sample Code

Appends a line at the end of the file.

```
$objTextFileParser = New-Object -ComObject
"NwSapSetupATLCommon.TextFileParser"
if ($objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services"))
{
    $objTextFileParser.AppendLine("spmsXYZ 75988/tcp")
    $objTextFileParser.Save("C:\Windows\System32\drivers\etc\services")
}
```

Reference

- `<$objTextFileParser>.Parse` [page 95]
- `<$objTextFileParser>.Save` [page 99]

<$objTextFileParser>.NumLines

Returns the number of lines in the text buffer of the `TextFileParser` object.

Syntax

```
<$objTextFileParser>.NumLines()
```

Returns

Returns `[long]` number of lines in the buffer.

Remarks

If a file was parsed `NumLines()` returns the number of lines read from the file. If no file was parsed `NumLines()` return `0`.

Sample Code

Checks the number of lines in buffer and saves to disk.

```
$objTextFileParser = New-Object -ComObject
"NwSapSetupATLCommon.TextFileParser"
if ($objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services"))
{
    $NwEngine.Context.Log.Write("Services file has " +
$objTextFileParser.NumLines() + " lines.")
}
```

Reference

- `<$objTextFileParser>.Parse` [page 95]
- `$NwEngine.Context.Log.Write` [page 90]

<$objTextFileParser>.Line

Returns the line indicated by the given index.

Syntax

```
<$objTextFileParser>.Line([long]$index)
```

Arguments

`index` required: Number of the line in the buffer.

Returns

Returns `[string]` content of the line indicated by the given index.

Remarks

The first line of the buffer is indicated by `1`. This function fails if a line outside the buffer is accessed. Use `NumLines()` in the first step to get the number of lines in the buffer.

Sample Code

```
$objTextFileParser = New-Object -ComObject
"NwSapSetupATLCommon.TextFileParser"
if ($objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services") -and
$objTextFileParser.NumLines() -gt 0)
{
    $line = $objTextFileParser.Line(1)
    $NwEngine.Context.Log.Write("Line 1: " + line)
}
```

Reference

- `<$objTextFileParser>.Parse` [page 95]

<$objTextFileParser>.RemoveLine

Removes a line from the text buffer of the `TextFileParser` object.

Syntax

```
<$objTextFileParser>.RemoveLine([string]$line)
```

Arguments

`line` required. Text to be removed from the buffer.

Returns

This method does not return anything.

Remarks

`RemoveLine` removes text from the internal buffer of the `TextFileParser`.

Sample Code

```
$objTextFileParser = New-Object -ComObject
"NwSapSetupATLCommon.TextFileParser"
if ($objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services") -and
$objTextFileParser.NumLines() -gt 0)
{
    if ($objTextFileParser.DoesLineExist("sapgw99.  3399/tcp"))
    {
        $objTextFileParser.RemoveLine("sapgw99.  3399/tcp")
        $objTextFileParser.Save("C:\Windows\System32\drivers\etc\services")
    }
}
```

Reference

- `<$objTextFileParser>.Parse` [page 95]
- `<$objTextFileParser>.DoesLineExist` [page 98]
- `<$objTextFileParser>.Save` [page 99]

8.2 Visual Basic Script Application Program Interfaces

Packages can be extended by package event scripts. These are written in Visual Basic Script. SAPSetup provides Visual Basic Script objects for easy scripting:

- **NwEngine.Shell**
  Handling of files and directories, Windows registry entries, Executing and controlling processes
- **NwEngine.Context**
  Checking the context of the installation, like Windows version, hostname, command line arguments
- **NwEngine.Context.Log**
  Writing log file entries

- **NwEngine.Variables**
  Getting values of SAPSetup variables

All package information, including package event scripts, is stored in the file `Setup/SapPackageSetup.xml`. Make sure to backup your installation server and especially this file, as SAPSetup does not provide a history of package versions. Thus when you change a package, the previous version is gone. When you change or delete a package event script, the previous version is gone.

For basics on Visual Basic Script check out

- VBScript Language Reference on MSDN at https://msdn.microsoft.com
- VBScript Fundamentals on MSDN at https://msdn.microsoft.com

Testing Scripts

Make sure to test your scripts. Try out how it behaves if a network path is not available or some file is not there as expected. Test all life cycle steps: install, update and uninstall.

Have your script fail gracefully. Several methods return a success indication, so you can react on it. Writing an error to the log file will make SAPSetup return a value other than 0. Thus you can make your infrastructure react accordingly, like have the installation, update or uninstall repeated.

Include Custom Files in Packages

Additional files can be included in packages. Create a directory named `CustomerFiles` in the root directory of your installation server and copy your files into this directory. All files and directories in this directory will be included in every package single-file installer. Use package event scripts to install your custom files.

Sample Code

```
'Use this in package event script "On Installation End"
NwEngine.Shell.CopyFile "%SapSrcDir%\CustomerFiles\myFile.txt",
"%COMMONPROGRAMFILES64%\myFile.txt"
'Use this in package event script "On Update End"
'when you do something during installation of the pacakge, always consider
what should happen when updating the package.
NwEngine.Shell.CopyFileEx "%SapSrcDir%\CustomerFiles\myFile.txt",
"%COMMONPROGRAMFILES64%\myFile.txt", vbFalse
'Use this in package event script "On Uninstallation End"
'when you install something, clean up during uninstall.
NwEngine.Shell.DeleteFile "%COMMONPROGRAMFILES64%\myFile.txt"
```

Installs the file `myFile.txt` into directory `%COMMONPROGRAMFILES64%\myProduct`. When the package is updated, this file will be updated as well, overwriting the existing file. During uninstall of the package, the file will be deleted.

Procedures Versus Functions

Visual Basic script distinguishes procedures and functions. Procedures do not return a value, and their parameters must not be enclosed by parantheses. Functions do return a value, and their parameters must be enclosed by parantheses. Almost all methods of NwEngine are functions, except those for logging and very few others.

Sample Code

```
'Call a procedure
'Leave the parameters without paranthesis, else the script will fail with
syntax error.
NwEngine.Context.Log.Write "Some text"
'Call a function
'Enclose the parameters with paranthesis.
Dim result
Result = NwEngine.Shell.CopyFile("%SapSrcDir%\CustomerFiles\myFile.txt",
"%COMMONPROGRAMFILES64%\myFile.txt")
```

Line Wrapping and Text Concatenation

Long lines can be wrapped, and strings concatenated.

Sample Code

```
'Without Wrapping
Dim Text
Text = "This is a very long line. It is to long to fit in an editor window so
you must scroll horizontal to see all the content"
'With Wrapping and Concatenation
Text = "This is a very long line. It is to long to fit in an editor" & _
       " window so you must scroll horizontal to see all the content"
'With the "_" character you can wrap all code lines
If NwEngine.Context.Windows8 = vbFalse And _
   NwEngine.Context.Windows81 = vbFalse Then
'    Do something
End If
```

8.2.1 NwEngine.Shell

`NwEngine.Shell` comprises methods to access and manipulate the file system and the Windows registry.

Methods for File System Access

Method Name	Description
`CopyDirectory(String source, String target)` [page 107]	Copies source directory to target directory with all content. Returns true if copying was successful, else false.
`CopyDirectoryEx(String source, String target, Bool failIfExists)` [page 108]	Copies source directory to target directory with all content. If `failIfExists` is set to true, existing files will not be overwritten. Returns true if copying was successful, else false.
`CopyFile(String source, String target)` [page 109]	Copies source file to target file on the file system. Returns true if deletion was successful, else false.
`CopyFileEx(String source, String target, Bool failIfExists)` [page 109]	Copies source file to target file on the file system. If `failIfExists` is set to true, an existing file will not be overwritten. Returns true if deletion was successful, else false.
`CreateDirectory(String directory)` [page 110]	Creates a directory. Returns true if creation was successful, else false.
`CreateTempDirectory()` [page 111]	Creates a temporary directory and returns the path as string.
`DeleteDirectory(String directory)` [page 111]	Deletes a directory. Returns true if deletion was successful, else false.
`DeleteFile(String file)` [page 112]	Deletes a file. Returns true if deletion was successful, else false.
`DirectoryExist(String path)` [page 114]	Returns true if the directory exists, else false.
`FileExist(String file)` [page 118]	Returns true if the file exists, else false.
`GetDriveType(String drive)` [page 119]	Returns a value indicating the type of the drive as integer.
`GetFileExt(String file)` [page 119]	Returns the extension of a file name as string.
`GetFileName(String path)` [page 120]	Returns the file name of the path as string.

Method Name	Description
`GetFileSize(String file)` [page 120]	Returns the size of the file in bytes as long.
`GetFileVersion(String file)` [page 121]	Returns the version number of the file as string.
`GetPath(String file)` [page 121]	Returns the path of a file as string.
`GetShortPath(String file)` [page 123]	Returns the short path in 8.3 format of a file as string.
`IsPathEqual(String path1, String path2)` [page 124]	Compares the pathes of two directories and returns true if they are identical, else false.

Methods for Process and System Interaction

Method Name	Description
`Execute(String cmdline, bool async)` [page 116]	Executes a command line. Returns true if the process has finished, else false.
`ExecuteEx(String cmdline, bool async, bool visible)` [page 117]	Executes a command line, optinally without displaying a window. Returns true if the process has finished, else false.
`ExecuteRC(String cmdline, bool visible)` [page 118]	Executes a command line, optinally without displaying a window. Returns the return code of the process, if executed synchronously.
`GetSystemUtcTime()` [page 123]	Returns the system UTC time as string.
`IsExeRunning(String file)` [page 124]	Returns true if the file is a running executable, else false.
`Sleep Number milliseconds` [page 128]	Make SAPSetup sleep for the specified time.
`TerminateProcessInstances String exeName` [page 128]	Terminates all processes whose executable name is exeName.

Methods for Registry Access

Method Name	Description

> **Note**
>
> On 64-bit Windows, SAPsetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 129] property.

`DeleteRegKey(String key)` [page 112]	Deletes a registry key and returns true if deletion was successful, else false.
`DeleteRegValue(String value, String valueName)` [page 113]	Deletes a registry value and returns true if deletion was successful, else false.
`EnumRegKeys(String path)` [page 115]	Returns a collection containing all keys below the given path.
`EnumRegValues(String path)` [page 116]	Returns a collection containing all values below the given path.
`GetRegValue(String key,String valueName)` [page 122]	Returns the data of the registry value.
`RegKeyExist(String key)` [page 125]	Returns true if registry key exists, else false.
`RegValueExist(String value, String valueName)` [page 126]	Returns true if registry value exists, else false.
`SetRegValue(String valuePath, String valueType, String value, String valueName)` [page 127]	Sets a registry value.

Properties

Property Name	Description
`Use64BitHive` [page 129]	Gets or sets a flag indicating whether SAPSetup operates on the 64bit branch of the Windows registry.

NwEngine.Shell.CopyDirectory

Copies source directory to target directory with all content.

Syntax

```
NwEngine.Shell.CopyDirectory(String sourceDir, String target)
```

Arguments

`sourceDir` required. The path to the directory to copy.

`target` required. If the target is a directory, the target file name is the same as the source file name. If the target does not exist, or if it is a file, then this is the target file name.

Returns

Returns `Bool: true` if the directory copy was successful, else it returns `Bool: false`.

Remarks

Existing files will not be overwritten.

Sample Code

```
Dim source : source = NwEngine.Variables.ResolveString("%SapSrcDir%
\CustomerFiles\myfiles")
If NwEngine.Shell.CopyDirectory(source , "C:\path\to\yourfiles") Then
    NwEngine.Context.Log.Write "Copying %SapSrcDir%\CustomerFiles\myfiles to
C:\path\to\yourfiles was successful."
End If
```

Copies the directory `%SapSrcDir%\CustomerFiles\myfiles` to `C:\path\to\yourfiles` and writes a message to the log, if the directory copy succeeded.

NwEngine.Shell.CopyDirectoryEx

Copies source directory to target directory with all content.

Syntax

```
NwEngine.Shell.CopyDirectoryEx(String sourceDir, String target, Bool overwrite)
```

Arguments

`sourceDir` required. The path to the directory to copy.

`target` required. If the target is a directory, the target file name is the same as the source file name. If the target does not exist, or if it is a file, then this is the target file name.

`overwrite` required. If set to `vbTrue`, an existing file will be overwritten. If set to `vbFalse`, existing files will not be overwritten.

Returns

Returns `Bool: true` if the directory copy was successful, else it returns `Bool: false`.

Remarks

Copies source directory to target directory with all content. When setting `noOverwrite` to `Bool: true`, this method behaves like `NwEngine.Shell.CopyDirectory()`. This method is useful to update files copied from `CustomerFiles` on the installation server (see Creating and Deploying a Single-File Installer for Packages [page 28]).

Sample Code

```
srcFolder = NwEngine.Variables.ResolveString("%SapSrcDir%
\CustomerFiles\myfiles")
dstFolder = NwEngine.Variables.ResolveString("C:\path\to\yourfiles")
If NwEngine.Shell.CopyDirectoryEx(srcFolder , dstFolder , vbFalse) Then
    NwEngine.Context.Log.Write "Copying %SapSrcDir%\CustomerFiles\myfiles to
C:\path\to\yourfiles was successful."
End If
```

Copies the directory `%SapSrcDir%\CustomerFiles\myfiles` to `C:\path\to\yourfiles`, overwriting existing files, and writes a message to the log, if the directory copy succeeded.

NwEngine.Shell.CopyFile

Copies source file to target file on the file system.

Syntax

`NwEngine.Shell.CopyFile(String sourceFile, String target)`

Arguments

`sourceFile` required. The name of the file to copy.

`target` required. If the target is a directory, the target file name is the same as the source file name. If the target does not exist, or if it is a file, then this is the target file name.

Returns

Returns `Bool: true` if file copy was successful, else it returns `Bool: false`.

Remarks

Existing files will not be overwritten.

Sample Code

```
If NwEngine.Shell.CopyFile("%SapSrcDir%\CustomerFiles\BrandingImage.png",
"%SapFrontEndDir%\SapGui\BrandingImage.png") Then
    NwEngine.Context.Log.Write "Copying %SapSrcDir%
\CustomerFiles\BrandingImage.png to folder %SapFrontEndDir%\SapGui was
successful"
End If
```

Copies the file `%SapSrcDir%\setup\netstart.exe` to directory `C:\path\to\setup` and writes a message to the log, if the file copy succeeded.

NwEngine.Shell.CopyFileEx

Copies source file to target file on the file system.

Syntax

```
NwEngine.Shell.CopyFileEx(String sourceFile, String target, Bool overwrite)
```

Arguments

`sourceFile` required. The name of the file to copy.

`target` required. If the target is a directory, the target file name is the same as the source file name. If the target does not exist, or if it is a file, then this is the target file name.

`overwrite` required. If set to `vbTrue`, an existing file will be overwritten. If set to `vbFalse`, existing files will not be overwritten.

Returns

Returns `Bool: true` if file copy was successful, else it returns `Bool: false`.

Remarks

When setting `noOverwrite` to `Bool: true`, this method behaves like `NwEngine.Shell.CopyFile()`. This method is useful to update files copied from `CustomerFiles` on the installation server (see Creating and Deploying a Single-File Installer for Packages [page 28].

Sample Code

```
If NwEngine.Shell.CopyFileEx("%SapSrcDir%\CustomerFiles\BrandingImage.png",
"%SapFrontEndDir%\SapGui\BrandingImage.png", vbTrue) Then
    NwEngine.Context.Log.Write "Copying %SapSrcDir%
\CustomerFiles\BrandingImage.png to folder %SapFrontEndDir%\SapGui was
successful"
End if
```

Copies the file `%SapSrcDir%\setup\netstart.exe` to directory `C:\path\to\setup` and writes a message to the log, if the file copy succeeded. If the file exists at the destination, it will be overwritten.

NwEngine.Shell.CreateDirectory

Creates a directory.

Syntax

```
NwEngine.Shell.CreateDirectory(String target)
```

Arguments

`target` required. Path to the directory to create.

Returns

Returns `Bool: true` if the creation was successful, else it returns `Bool: false`.

Sample Code

```
If Not NwEngine.Shell.CreateDirectory("C:\path\to\myDir") Then
```

```
    NwEngine.Context.Log.WriteError "Creating directory C:\path\to\myDir
failed."
End if
```

Tries to create directory `C:\path\to\myDir` and writes the error message `"Creating directory C:\path\to\myDir failed."` to the log, if the creation failed.

NwEngine.Shell.CreateTempDirectory

Creates a temporary directory.

Syntax

`NwEngine.Shell.CreateTempDirectory()`

Returns

Returns the path of the created directory as `String`.

Remarks

The temporary directory is created in `%TEMP%`. This is useful for more complex file operations that should not be done within the destination directory of the installation.

Sample Code

```
Dim tempPath
tempPath = NwEngine.Shell.CreateTempDirectory()
'Do some file operations there
NwEngine.Shell.DeleteDirectory(tempPath)
```

Creates a directory in `%TEMP%` and finally deletes it again.

NwEngine.Shell.DeleteDirectory

Deletes a directory.

Syntax

`NwEngine.Shell.DeleteDirectory(String target)`

Arguments

`target` required. Path to the directory to delete.

Returns

Returns `Bool: true` if the deletion was successful, else it returns `Bool: false`.

Remarks

> ⚠ Caution
>
> All content of the directory will be deleted. Use with great care.

> {·} Sample Code

```
Dim tempPath
tempPath = NwEngine.Shell.CreateTempDirectory()
'Do stuff in the temporary directory, then clean it up.
NwEngine.Shell.DeleteDirectory(tempPath)
```

Creates a directory in `%TEMP%` and returns the path to it. Afterwards the temporary directory is deleted.

NwEngine.Shell.DeleteFile

Deletes a file.

Syntax

`NwEngine.Shell.DeleteFile(String target)`

Arguments

`target` required. Path to the file to delete.

Returns

Returns `Bool: true` if the deletion was successful, else it returns `Bool: false`.

> {·} Sample Code

```
Dim someFile
someFile = "C:\path\to\myFile.ext"
NwEngine.Shell.DeleteFile(someFile)
```

Deletes the file `C:\path\to\myFile.ext`.

NwEngine.Shell.DeleteRegKey

> ⓘ Note
>
> On 64-bit Windows, SAPsetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 129] property.

Deletes a registry key.

Syntax

```
NwEngine.Shell.DeleteRegKey(String key)
```

Arguments

`key` required. Path to the registry key to delete.

Returns

Returns `Bool: true` if the deletion was successful, else it returns `Bool: false`.

Sample Code

```
Dim key1
key1 = "Internet Explorer\ActiveX Compatibility\{C0A63B86-4B21-11D3-BD95-
D426EF2C7949}"
NwEngine.Context.Log.Write "Removing the killbit for vsflex7l.ocx."
If NwEngine.Shell.RegKeyExist("HKLM\SOFTWARE\Microsoft\" & key1) Then
    If Not NwEngine.Shell.DeleteRegKey("HKLM\SOFTWARE\Microsoft\" & key1) Then
        NwEngine.Context.Log.Write "Deleting the killbit for vsflex7l.ocx has
failed."
    Else
        NwEngine.Context.Log.Write "Deleting the killbit for vsflex7l.ocx has
succeeded."
    End If
End If
```

Removes the killbit of `vsflex71.ocx` by deleting the corresponding registry key. Writes appropriate messages to the log file.

NwEngine.Shell.DeleteRegValue

Note

On 64-bit Windows, SAPsetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 129] property.

Deletes a registry value.

Syntax

```
NwEngine.Shell.DeleteRegValue(String value, String valueName)
```

Arguments

`value` required. Path to the registry value to delete.

`valueName` optional. Name of the registry value to delete.

Returns

Returns `Bool: true` if the deletion was successful, else it returns `Bool: false`.

Sample Code

```
Dim value
value =
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\SunJavaUpdateSched"
NwEngine.Context.Log.Write "Removing autostart of Java Update Scheduler."
If Not NwEngine.Shell.DeleteRegValue(value) Then
    NwEngine.Context.Log.Write "Deleting the autostart of Java Update
Scheduler has failed."
Else
    NwEngine.Context.Log.Write "Deleting the autostart of Java Update
Scheduler has succeeded."
End If
```

Removes the autostart of Java Update Scheduler by deleting the corresponding registry value. Writes appropriate messages to the log file.

Sample Code

```
Dim value, valueName
value = "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs"
valueName = "C:\WINDOWS\system32\mfc140.dll"
NwEngine.Context.Log.Write "Removing SharedDll counter for
C:\WINDOWS\system32\mfc140.dll."
If Not NwEngine.Shell.DeleteRegValue(value, valueName) Then
    NwEngine.Context.Log.Write "Deleting the registry value for
C:\WINDOWS\system32\mfc140.dll failed."
Else
    NwEngine.Context.Log.Write "Deleting the registry value
C:\WINDOWS\system32\mfc140.dll has succeeded."
End If
```

Optional parameter is required to delete a registry value which contains a backslash (‘\’). Otherwise `DeleteRegValue` tries to split registry key and value with the last backslash and will fail as the registry key cannot be found.

NwEngine.Shell.DirectoryExist

Checks if a directory exists.

Syntax

`NwEngine.Shell.DirectoryExist(String target)`

Arguments

`target` required. Path to the directory to check for existence.

Returns

Returns `Bool: true` if the directory exists, else it returns `Bool: false`.

Sample Code

```
Dim someDir
someDir = "C:\path\to\myDir"
If NwEngine.Shell.DirectoryExist(someDir) Then
    NwEngine.Context.Log.Write "File " & someDir & " exists."
End If
```

Writes a line to the log if the file `C:\path\to\myDir` exists.

NwEngine.Shell.EnumRegKeys

Note

On 64-bit Windows, SAPsetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 129] property.

Returns a collection containing all keys below a specified registry path.

Syntax

```
NwEngine.Shell.EnumRegKeys(String path)
```

Arguments

`path` required. Path to the registry key to enumerate.

Returns

Returns a `collection` of the registry keys that are direct children of the specified path. Does not return the whole tree below the specified path recursively.

Sample Code

```
Dim jre8Version
If NwEngine.Shell.RegKeyExist("HKLM\SOFTWARE\JavaSoft\Java Runtime
Environment\1.8") Then
    For Each jre8Key in
NwEngine.Shell.EnumRegKeys("HKLM\SOFTWARE\JavaSoft\Java Runtime Environment")
        If Left(jre8Key, 5) = "1.8.0" Then
            jre8Version = jre8Key
        End If
    Next
End If
```

NwEngine.Shell.EnumRegValues

> **Note**
>
> On 64-bit Windows, SAPsetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 129] property.

Returns a collection containing all values below a specified registry path.

Syntax

```
NwEngine.Shell.EnumRegValues(String path)
```

Arguments

`path` required. Path to the registry key to enumerate.

Returns

Returns a `collection` of the registry values that are direct children of the specified path. Does not return the whole tree below the specified path recursively.

> **Sample Code**

```
Dim autoRunPath
autoRunPath = "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
For Each autoRunValue in NwEngine.Shell.EnumRegValues(autoRunPath)
    NwEngine.Context.Log.Write "autoRunValue : " & autoRunValue
Next
'Do not forget the 64 bit registry hive
NwEngine.Shell.Use64BitHive = vbTrue
For Each autoRunValue in NwEngine.Shell.EnumRegValues(autoRunPath)
    NwEngine.Context.Log.Write "autoRunValue : " & autoRunValue
Next
'Always reset to the 32bit registry hive
NwEngine.Shell.Use64BitHive = vbfalse
```

Enumerates all registry values in `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` in both, the 32bit hive and the 64bit hive. Finally resets the registry mode back to 32bit hive.

NwEngine.Shell.Execute

Executes a command line.

Syntax

```
NwEngine.Shell.Execute(String cmdline, Bool async)
```

Arguments

`cmdline` required. Command line to execute.

`async` required. If async is set to `Bool: true`, the process will be started asynchronously. If async is set to `Bool: false`, SAPSetup waits for the process to finish.

Returns

Returns `Bool: true`true if the process has finished, else `Bool: false`.

Sample Code

```
Dim cmdline
cmdline = "notepad.exe C:\path\to\readme.txt"
If NOT NwEngine.Context.Silent Then
    NwEngine.Context.Log.Write "Showing readme file."
    NwEngine.Shell.Execute cmdline, vbTrue
End If
```

If Microsoft Word is installed, and SAPSetup is not running silently, then it is called to show the file `C:\path\to\readme.doc` exists. Microsoft Word is started asynchronously, so that the installation can finish. Before calling command lines that require interaction, check if SAPSetup is running interactively.

NwEngine.Shell.ExecuteEx

Executes a command line, optinally without displaying a window.

Syntax

```
NwEngine.Shell.ExecuteEx(String cmdline, Bool async, Bool visible)
```

Arguments

`cmdline` required. Command line to execute.

`async` required. If async is set to true, the process will be started asynchronously. If async is set to false, SAPSetup waits for the process to finish.

`visible` required. If visible is set to false, no window will be shown. This is especially useful for executing batch processes.

Returns

Returns `Bool: true`true if the process has finished, else `Bool: false`.

Sample Code

```
Dim cmdline
cmdline = "%SapSrcDir%\CustomerFiles\mybatch.cmd"
NwEngine.Context.Log.Write "Executing mybatch.cmd ..."
Dim retVal : retVal = NwEngine.Shell.ExecuteEx(cmdline, vbFalse, vbTrue)
```

Executes `%SapSrcDir%\CustomerFiles\mybatch.cmd` without showing the command prompt window. Waits for the execution before continuing.

NwEngine.Shell.ExecuteRc

Executes a command line, optinally without displaying a window.

Syntax

`NwEngine.Shell.ExecuteRC(String cmdline, Bool visible)`

Arguments

`cmdline` required. Command line to execute.

`visible` required. If visible is set to false, no window will be shown. This is especially useful for executing batch processes.

Returns

Returns the return code of the executed command as `String`.

Sample Code

```
Dim cmdline, returnCode
cmdline = "%SapSrcDir%\CustomerFiles\mybatch.cmd"
NwEngine.Context.Log.Write "Executing mybatch.cmd ..."
returnCode = NwEngine.Shell.ExecuteRC(cmdline, vbTrue)
NwEngine.Context.Log.Write "Executing mybatch.cmd finished with " & returnCode
```

Executes `%SapSrcDir%\CustomerFiles\mybatch.cmd` without showing the command prompt window. Waits for the execution before continuing, and prints the return code of the executed command to the log file.

NwEngine.Shell.FileExist

Checks if a file exists.

Syntax

`NwEngine.Shell.FileExist(String target)`

Arguments

`target` required. Path to the file to check for existence.

Returns

Returns `Bool: true` if the file exists, else it returns `Bool: false`.

Sample Code

```
Dim someFile
someFile = "C:\path\to\myFile.ext"
If NwEngine.Shell.FileExist(someFile) Then
    NwEngine.Context.Log.Write "File " & someFile & " exists."
End If
```

Writes a line to the log if the file `C:\path\to\myFile.ext` exists.

NwEngine.Shell.GetDriveType

Returns the type of a drive.

Syntax

```
NwEngine.Shell.GetDriveType(String drive)
```

Arguments

`drive` required. Path to the drive to check.

Returns

Returns a `Number` indicating the drive type:

- 0: DriveTypeFixed (e.g. HDD)
- 1: DriveTypeRemovable (e.g. floppy)
- 2: DriveTypeRemote (e.g. network drive)
- 3: DriveTypeCDROM (e.g. DVD-ROM)
- 64: DriveTypeUnknown

Sample Code

```
NwEngine.Context.Log.Write "Type of drive C: is: " &
NwEngine.Shell.GetDriveType("C:")
```

Writes the numeric type of drive `C:` to the log.

NwEngine.Shell.GetFileExt

Returns the extension of a file name.

Syntax

```
NwEngine.Shell.GetFileExt(String file)
```

Arguments

`file` required. File to get the extension of.

Returns

Returns the file extension of a file as `String`.

Sample Code

```
someFile = "C:\path\to\myFile.ext"
NwEngine.Context.Log.Write "Extension of file " & someFile & " is "&
NwEngine.Shell.GetFileExt(someFile)
```

Writes the file extension of file `someFile` to the log.

NwEngine.Shell.GetFileName

Returns the file extension of a file.

Syntax

```
NwEngine.Shell.GetFileName(String path)
```

Arguments

`path` required. Path to get the filename of.

Returns

Returns the file name of a path as `String`.

Sample Code

```
dim sapsetup
sapsetup = NwEngine.Variables.ResolveString("%SAPSRCDIR%
\Setup\NwSapSetup.exe")
NwEngine.Context.Log.Write "Filename of SAPSetup is " &
NwEngine.Shell.GetFileName(sapsetup)
```

Writes the file name of the SAPSetup executable to the log.

NwEngine.Shell.GetFileSize

Returns the size of a file in bytes.

Syntax

```
NwEngine.Shell.GetFileSize(String file)
```

Arguments

`file` required. File to get the size of.

Returns

Returns the size of a file as `Number` in bytes.

Sample Code

```
Dim myfile, size
myfile = NwEngine.Variables.ResolveString("%SapSrcDir%
\CustomerFiles\myfile.txt")
size = NwEngine.Shell.GetFileSize(myfile)
If size = 0 Then
    NwEngine.Context.Log.Write "myfile.txt is empty"
End If
```

Checks the size of file `%SapSrcDir%\CustomerFiles\myfile.txt` and writes an info to the log, if the file has size 0.

NwEngine.Shell.GetFileVersion

Returns the file version of a file.

Syntax

```
NwEngine.Shell.GetFileVersion(String file)
```

Arguments

`file` required. File to get the version of.

Returns

Returns the file version of a file as `String`.

Sample Code

```
Dim cmdExe
cmdExe = NwEngine.Variables.ResolveString("%WINSYSDIR64%\cmd.exe")
NwEngine.Context.Log.Write "Version of cmd.exe is " &
NwEngine.Shell.GetFileVersion(cmdExe)
```

Resolves the path to the 64bit `cmd.exe` . Writes the file version of file `cmd.exe` to the log.

NwEngine.Shell.GetPath

Returns the path of a file.

Syntax

```
NwEngine.Shell.GetPath(String file)
```

Arguments

`file` required. File to get the path of.

Returns

Returns the path to a file as `String`.

Sample Code

```
Dim cmdExe
cmdExe = NwEngine.Variables.ResolveString("%WINSYSDIR64%\cmd.exe")
NwEngine.Context.Log.Write "Path to cmd.exe is " &
NwEngine.Shell.GetPath(cmdExe)
```

Resolves the path to the 64bit `cmd.exe`. Writes the path to file `cmd.exe` to the log.

NwEngine.Shell.GetRegValue

Note

On 64-bit Windows, SAPsetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 129] property.

Returns the value of a registry key.

Syntax

```
NwEngine.Shell.GetRegValue(String key, String valueName)
```

Arguments

`key` required. Registry key to get the value of.

`valueName` optional. Name of the registry value to get.

Returns

Returns the value of a registry key as `String`.

Sample Code

```
If NwEngine.Shell.RegValueExist("HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon") Then
    Dim RegVal
    RegVal =
NwEngine.Shell.GetRegValue("HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon")
    NwEngine.Context.Log.Write "Registry value of " & _
"HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon is: " & RegVal
End if
```

If the key `HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon` exists, its value is written to the log.

Sample Code

```
If
NwEngine.Shell.RegValueExist("HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\S
haredDLLs", "C:\WINDOWS\system32\mfc140.dll") Then
    Dim RegVal
    RegVal =
NwEngine.Shell.GetRegValue("HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Sha
redDLLs", "C:\WINDOWS\system32\mfc140.dll")
    NwEngine.Context.Log.Write "Registry value of
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs\C:\WINDOWS\system32\
mfc140.dll is: " & RegVal
End if
```

Optional parameter is required to retrieve a registry value which contains a backslash (‘\’). Otherwise `GetRegValue` tries to split registry key and value with the last backslash and will fail as the registry key cannot be found.

NwEngine.Shell.GetShortPath

Returns the short path in 8.3 notation of a path.

Syntax

`NwEngine.Shell.GetShortPath(String path)`

Arguments

`path` required. Path to get the short path of.

Returns

Returns the short path in 8.3 notation of a path as `String`.

Sample Code

```
Dim customerfiles
customerfiles = NwEngine.Variables.ResolveString("%SapSrcDir%\CustomerFiles")
NwEngine.Context.Log.Write "Short path to CustomerFiles is " &
NwEngine.Shell.GetShortPath(customerfiles)
```

Resolves the path to the directory `CustomerFiles` on the installation server. Writes the path to it to the log.

NwEngine.Shell.GetSystemUtcTime

Returns the system UTC time.

Syntax

`NwEngine.Shell.GetSystemUtcTime()`

Returns

Returns the system UTC time as `String`.

Sample Code

```
NwEngine.Context.Log.Write "Current UTC time is " &
NwEngine.Shell.GetSystemUtcTime()
```

Writes the current system UTC time to the log.

NwEngine.Shell.IsExeRunning

Checks if a specific executable is running.

Syntax

`NwEngine.Shell.IsExeRunning(String filename)`

Arguments

`filename` required. File that is checked if it is running as executable.

Returns

Returns `Bool: true` if the file is running as executable.

Sample Code

```
If NwEngine.Shell.IsExeRunning("saplogon.exe") Then
    NwEngine.Context.Log.Write "SAPlogon is still running."
End If
```

Writes a line to the log if `saplogon.exe` is running.

NwEngine.Shell.IsPathEqual

Compares two pathes for equality.

Syntax

`NwEngine.Shell.IsPathEqual(String path1, String path2)`

Arguments

`path1` required. First path to compare.

`path2` required. Second path to compare.

Returns

Returns `Bool: true` if the two pathes point to the same location, otherwise returns `Bool: false`.

Remarks

This method is helpful to compare the following:

- Paths to the network
- Paths that contain environment variables
- Absolute and relative paths

Sample Code

```
Dim pathFromRegistry, pathOnDisk
pathFromRegistry =
NWEngine.Shell.GetRegValue("HKLM\SOFTWARE\MySoftware\MyKey\MyPath")
pathOnDisk = NwEngine.Variables.ResolveString("%ProgramFiles%
\MySoftware\MyFile.ext")
If NwEngine.Shell.IsPathEqual(pathFromRegistry, pathOnDisk) Then
    NwEngine.Context.Log.Write "Path in registry is same as " & pathOnDisk
End If
```

Retrieves the value `HKLM\SOFTWARE\MySoftware\MyKey\MyPath` from the Windows registry and compares it to the path `%ProgramFiles%\MySoftware\MyFile.ext`. If the two paths point to the same location, writes an entry to the log.

NwEngine.Shell.RegKeyExist

Note

On 64-bit Windows, SAPsetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 129] property.

Checks if a registry key exists.

Syntax

```
NwEngine.Shell.RegKeyExist(String key)
```

Arguments

`key` required. Path to the registry key to check.

Returns

Returns `Bool: true` if the registry key exists, else it returns `Bool: false`.

Sample Code

```
If NwEngine.Shell.RegKeyExist("HKLM\SOFTWARE\SAP") Then
    NwEngine.Context.Log.Write "Registry key HKLM\SOFTWARE\SAP exists"
End if
```

Writes a line to the log if registry key `HKLM\SOFTWARE\SAP` exists.

NwEngine.Shell.RegValueExist

> **Note**
>
> On 64-bit Windows, SAPsetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 129] property.

Checks if a registry value exists.

Syntax

```
NwEngine.Shell.RegValueExist(String value, String valueName)
```

Arguments

`value` required. Path to the registry value to check.

`valueName` optional. Name of the registry value to check.

Returns

Returns `Bool: true` if the registry value exists, else it returns `Bool: false`.

> **Sample Code**

```
If NwEngine.Shell.RegValueExist("HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon") Then
    NwEngine.Context.Log.Write "Registry value
HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon exists"
End if
```

Writes a line to the log if registry value `HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon` exists.

> **Sample Code**

```
If
NwEngine.Shell.RegValueExist("HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\S
haredDLLs", "C:\WINDOWS\system32\mfc140.dll")Then
     NwEngine.Context.Log.Write "Registry value
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs\C:\WINDOWS\system32\
mfc140.dll exists"
End if
```

Optional parameter is required to check the existence of a registry value which contains a backslash (‘\’) . Otherwise `RegValueExist` tries to split registry key and value with the last backslash and will fail as the registry key cannot be found.

NwEngine.Shell.SetRegValue

> **Note**
>
> On 64-bit Windows, SAPsetup writes registry entries to the `WOW6432Node` registry tree (32-bit registry hive) by default. When you use a key like `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\...` it is written to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\SAP\...` in the registry. It should never be necessary to specify `WOW6432Node` in registry script commands. If you want to create entries in the 64-bit registry tree, use the `Use64BitHive` [page 129] property.

Sets a registry value.

Syntax

```
NwEngine.Shell.SetRegValue(String valuePath, String valueType, String value, String valueName)
```

Arguments

`valuePath` required. Path to the registry value to set.

`valueType` required. Type of the registry value to set. Must be one of `"REG_SZ"`, `"REG_EXPAND_SZ"`, `"REG_MULTI_SZ"`, `"REG_BINARY"`, `"REG_DWORD"`.

`value` required. Value to set.

`valueName` optional. Name of the registry value to set.

Returns

Returns `Bool: true` if the registry value was set successfully, else it returns `Bool: false`.

> **Sample Code**

```
NwEngine.Shell.SetRegValue("HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon",
"REG_DWORD", "1")
```

Sets the registry value `HKLM\SOFTWARE\SAP\SAPGUI\StartSaplogon` to type `REG_DWORD` and value `1`.

> **Sample Code**

```
If
NwEngine.Shell.SetRegValue("HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs", "REG_DWORD", "1", "C:\WINDOWS\system32\mfc140.dll") Then
    NwEngine.Context.Log.Write "Registry value successfully created."
End If
```

Optional parameter is required to set a registry value which contains a backslash (‘\’). Otherwise `SetRegValue` tries to split registry key and value with the last backslash and will fail as the registry key cannot be found.

NwEngine.Shell.Sleep

Makes SAPSetup sleep for a specified time.

Syntax

`NwEngine.Shell.Sleep Number milliseconds`

Arguments

`milliseconds` required. Number of milliseconds SAPSetup shall sleep.

Remarks

This is useful when the script does something that has side effects, like shutting down a Windows service, which needs a few seconds, and there's no possibility to wait for it. As this method does not return anything, its parameters must not be enclosed by paranthesis.

Sample Code

```
NwEngine.Shell.Sleep 5000
```

Makes SAPSetup sleep for 5 seconds.

NwEngine.Shell.TerminateProcessInstances

Terminates all running processes with a specific executable.

Syntax

`NwEngine.Shell.TerminateProcessInstances String filename`

Arguments

`filename` required. Name of the executable to terminate.

Remarks

This call will terminate all processes that run with the specified executable. Useful if there is definitely no other way to shut down a running process in a controlled way. Use with great care. As this method does not return anything, its parameters must not be enclosed by paranthesis.

Sample Code

```
If NwEngine.Shell.IsExeRunning("saplogon.exe") Then
    NwEngine.Context.Log.Write "SAPlogon is still running. Killing all
instances of it."
    NwEngine.Shell.TerminateProcessInstances "saplogon.exe"
End If
```

Writes a line to the log if `saplogon.exe` is running.

NwEngine.Shell.Use64BitHive

Gets or sets a flag indicating whether SAPSetup operates on the 64bit branch of the Windows registry.

Syntax

```
NwEngine.Shell.Use64BitHive as Bool
```

Remarks

SAPSetup is a 32bit application. By default all operations on the Windows registry are done on the 32bit hive. Use this propertey to access the 64bit hive. When `use64bit` is set to true, following registry operations will be done on the 64bit branch of the registry. When use64bit is set to false, following registry operations will be done on the 32bit branch of the registry.

> **Note**
>
> Always reset to 32bit hive afterwards.

> **Sample Code**

```
NwEngine.Shell.Use64BitHive = vbTrue
'Here be operations on the 64bit hive of the registry
'Always reset to 32bit hive!
NwEngine.Shell.Use64BitHive = vbFalse
```

Set the registry mode so that SAPSetup operates on the 64bit hive, and resets it again to the 32bit hive.

8.2.2 NwEngine.Context

`NwEngine.Context` provides access to properties to determine operating system, computer name, rights elevation status, command line arguments and more.

Properties

Property Name	Description
`Elevated [page 130]`	Gets a flag indicating whether this installer instance is executed with elevated rights from LSH
`LCID [page 130]`	Gets the system locale identifier of the current workstation
`Reboot [page 131]`	Gets or sets a flag indicating whether the reboot flag should be set
`Silent [page 131]`	Gets or sets a flag indicating whether the installation mode was started in silent mode
`WinVista [page 131]`	Gets a flag indicating whether running on a Windows Vista or Windows Server 2008 machine

Property Name	Description
Windows7 [page 132]	Gets a flag indicating whether running on a Windows 7 or Windows Server 2008 R2 machine
Windows8 [page 132]	Gets a flag indicating whether running on a Windows 8 or Windows Server 2012 machine
Windows81 [page 132]	Gets a flag indicating whether running on a Windows 8.1 or Windows Server 2012 R2 machine
Windows10 [page 133]	Gets a flag indicating whether running on a Windows 10 or Windows Server 2016, 2019 or 2022 machine
Windows11 [page 133]	Gets a flag indicating whether running on a Windows 11 or Windows Server 2025 machine
WkstaName [page 134]	Gets the NetBIOS name of the current workstation
WTSClient [page 134]	Gets a flag indicating whether running on a WTS Client
WTSServer [page 134]	Gets a flag indicating whether running on a WTS Server

NwEngine.Context.Elevated

Gets a flag indicating whether this installer instance is executed with elevated rights from LSH.

Syntax

```
NwEngine.Context.Elevated as Bool
```

Sample Code

```
If NwEngine.Context.Elevated Then
    NwEngine.Context.Log.Write "We are running with elevation."
End If
```

Writes an entry to the log, if running elevated.

NwEngine.Context.LCID

Gets the hosting workstation's system locale identifier.

Syntax

```
NwEngine.Context.LCID as String
```

Sample Code

```
NwEngine.Context.Log.Write "System locale identifier is " &
NwEngine.Context.LCID
```

Writes the system locale identifier to the log.

NwEngine.Context.Reboot

Gets or sets a flag indicating whether the engine's reboot flag should be set.

Syntax

```
NwEngine.Context.Reboot as Bool
```

Sample Code

```
NwEngine.Context.Reboot = vbTrue
NwEngine.Context.Log.Write "Reboot will be required."
```

The procedure will require a reboot after it has finished. Writes an according entry to the log.

NwEngine.Context.Silent

Gets or sets a value indicating whether this installation mode was started in silent mode.

Syntax

```
NwEngine.Context.Silent as Bool
```

Sample Code

```
If NwEngine.Context.Silent Then
    NwEngine.Context.Log.Write "We are running in silent mode, showing no
dialogs."
End If
```

Writes an entry to the log, if running in silent mode.

NwEngine.Context.WinVista

Gets a flag indicating whether running on a Windows Vista or Windows Server 2008 machine.

Syntax

```
NwEngine.Context.WinVista as Bool
```

Sample Code

```
If NwEngine.Context.WinVista Then
    NwEngine.Context.Log.Write "We are running on Windows Vista or Windows
Server 2008."
End If
```

Writes an entry to the log, if running on Windows Vista or Windows Server 2008.

NwEngine.Context.Windows7

Gets a flag indicating whether running on a Windows 7 or Windows Server 2008 R2 machine.

Syntax

```
NwEngine.Context.Windows7 as Bool
```

Sample Code

```
If NwEngine.Context.Windows7 Then
    NwEngine.Context.Log.Write "We are running on Windows 7 or Windows Server
2008 R2."
End If
```

Writes an entry to the log, if running on Windows 7 or Windows Server 2008 R2.

NwEngine.Context.Windows8

Gets a flag indicating whether running on a Windows 8 or Windows Server 2012 machine.

Syntax

```
NwEngine.Context.Windows8 as Bool
```

Sample Code

```
If NwEngine.Context.Windows8 Then
    NwEngine.Context.Log.Write "We are running on Windows 8 or Windows Server
2012."
End If
```

Writes an entry to the log, if running on Windows 8 or Windows Server 2012.

NwEngine.Context.Windows81

Gets a flag indicating whether running on a Windows 8.1 or Windows Server 2012 R2 machine.

Syntax

```
NwEngine.Context.Windows81 as Bool
```

Sample Code

```
If NwEngine.Context.Windows81 Then
    NwEngine.Context.Log.Write "We are running on Windows 8.1 or Windows
Server 2012 R2."
End If
```

Writes an entry to the log, if running on Windows 8.1 or Windows Server 2012 R2.

NwEngine.Context.Windows10

Gets a flag indicating whether running on a Windows 10 or Windows Server 2016, 2019 or 2022 machine.

Syntax

```
NwEngine.Context.Windows10 as Bool
```

Sample Code

```
If NwEngine.Context.Windows10 Then
    NwEngine.Context.Log.Write "We are running on Windows 10 or Windows
Server 2016, 2019 or 2022."
End If
```

Writes an entry to the log, if running on Windows 10 or Windows Server 2016, 2019 or 2022.

NwEngine.Context.Windows11

Gets a flag indicating whether running on a Windows 11 or Windows Server 2025 machine.

Syntax

```
NwEngine.Context.Windows11 as Bool
```

Sample Code

```
If NwEngine.Context.Windows11 Then
NwEngine.Context.Log.Write "We are running on Windows 11 or Windows
Server 2025."
End If
```

Writes an entry to the log, if running on Windows 11 or Windows Server 2025.

NwEngine.Context.WkstaName

Gets the NetBIOS name of the current workstation.

Syntax

```
NwEngine.Context.WkstaName as String
```

Sample Code

```
Dim workstationName
workstationName = NwEngine.Context.WkstaName
NwEngine.Context.Log.Write "Name of this workstation: " & workstationName
```

Writes `Name of this workstation: <workstation Name>` into the log file.

NwEngine.Context.WTSClient

Gets a flag indicating whether this installer instance is executed on a WTS Client.

Syntax

```
NwEngine.Context.WTSClient as Bool
```

Sample Code

```
If NwEngine.Context.WTSClient Then
    NwEngine.Context.Log.Write "We are running on Windows Terminal Service
Client."
End If
```

Writes an entry to the log, if running on Windows Terminal Service Client.

NwEngine.Context.WTSServer

Gets a flag indicating whether this installer instance is executed on a WTS Server.

Syntax

```
NwEngine.Context.WTSServer as Bool
```

Sample Code

```
If NwEngine.Context.WTSServer Then
    NwEngine.Context.Log.Write "We are running on Windows Terminal Service
Server."
End If
```

Writes an entry to the log, if running on Windows Terminal Service Server.

8.2.3 NwEngine.Context.Log

`NwEngine.Context.Log` writes information texts, warnings, or errors to the log.

Procedure Name	Description
`Write String message` [page 135]	Writes a message into the current log file
`WriteError String errorMessage` [page 135]	Writes a message marked as error into the log file. Entry is marked as "E". SAPSetup return code will be not 0. Installation procedure will be handled as not successfully finished.
`WriteWarning String warningMessage` [page 136]	Writes a message marked as warning into the log file. Entry is marked as "W".

NwEngine.Context.Log.Write

Writes a message to the log file. Entry is marked as "E". SAPSetup return code will be not 0. Installation procedure will be handled as not successfully finished.

Syntax

```
NwEngine.Context.Log.Write String message
```

Arguments

`message` required. The message to write to the log file.

Remarks

As this method does not return anything, its parameters must not be enclosed by paranthesis.

Sample Code

```
NwEngine.Context.Log.Write "Hello World!"
```

Writes `"Hello World!"` to the log file.

NwEngine.Context.Log.WriteError

Write an error message to the log file. Entry is marked as "W".

Syntax

```
NwEngine.Context.Log.WriteError(String errorMessage)
```

Arguments

`errorMessage` required. The error message to write to the log file.

Remarks

As this method does not return anything, its parameters must not be enclosed by paranthesis. Writes a message into the current log file as an error. Installation procedure will be handled as not successfully finished, and SAPSetup will exit with a return code other than 0. This is useful to check prerequisites and make the installation process fail if the prerequisites are not fulfilled, so that it can be run again.

Sample Code

```
If somethingIsWrong Then
    NwEngine.Context.Log.WriteError "We will fail!"
End If
```

Writes `"We will fail!"` to the log file, marked as error with "E".

NwEngine.Context.Log.WriteWarning

Write a warning message to the log file.

Syntax

```
NwEngine.Context.Log.WriteWarning(String warningMessage)
```

Arguments

`warningMessage` required. The warning message to write to the log file.

Remarks

As this method does not return anything, its parameters must not be enclosed by paranthesis. Writes a message into the current log file as a warning. This does not change the return code of SAPSetup. This is useful if you want to note important things down during the installation, and later analyze the log file of SAPSetup. End users will never see this message.

Sample Code

```
If somethingIsStrange Then
    NwEngine.Context.Log.WriteWarning "We might fail!"
End If
```

Writes `"We might fail!"` to the log file, marked as warning with "W".

8.2.4 NwEngine.Variables

`NwEngine.Variables` comprises methods to access variable values and resolve strings that contain variables.

Remarks

The variable `<%SapSrcDir%>` points to the root directory of the current installation source. Regardless if it is an installation server or a single-file installer of a package, below `<%SapSrcDir%>` you find the same directory structure. The directory `%SapSrcDir%\CustomerFiles` is always included in every package. It is useful if you want to distribute your custom files with packages.

The package configuration in `NwSapSetupAdmin` shows additional variables that are available in the package. You can set the values of variables for packages there, and you can access these variables in package event scripts. SAPSetup provides a comprehensive list of additional variables containing information, among others, about the following:

- System directories
- Operating system
- Installed .NET runtimes
- Installed Java runtimes
- Installed Microsoft products
- Installed Adobe products

For the full list of all variables start SAPSetup and cancel it on the first dialog. Check the log file [page 52], look for the line `Log Variables Collection`. Do this on a system where no SAPSetup products are installed, as installing products persists some variables on the computer.

Methods for Variable Resolution

Method Name	Description
`GetValue(variableName)` [page 137]	Returns the value of the variable.
`ResolveString(StringToResolve)` [page 138]	Resolves all variables contained in `toResolve` and returns the resolved string.

NwEngine.Variables.GetValue

Returns the value of a variable.

Syntax

```
NwEngine.Variables.GetValue(String variableName)
```

Arguments

`variableName` required. Name of the variable to get the value of.

Returns

The value of the variable.

Sample Code

```
dim winmode
winmode = NwEngine.Variables.GetValue("WOW64MODE")
If winmode Then
    NwEngine.Context.Log.Write "Value of variable WOW64MODE is " & winmode &
". That means this Windows is 64bit."
Else
    NwEngine.Context.Log.Write "Value of variable WOW64MODE is " & winmode &
". That means this Windows is 32bit."
End if
```

Gets the value of variable `<WOW64MODE>` and writes it to the log file.

NwEngine.Variables.ResolveString

Resolves all variables contained in `toResolve` and returns the resolved string.

Syntax

```
NwEngine.Variables.ResolveString(String stringToResolve)
```

Arguments

`stringToResolve` required. A text that may contain SAPSetup variables like `<%variableName%>`.

Returns

Returns the provided string with all SAPSetup variables replaced by their values.

Sample Code

```
dim toResolve, resolved
toResolve = "%WINSYSDIR%\drivers\etc\hosts"
resolved = NwEngine.Variables.ResolveString(toResolve)
NwEngine.Context.Log.Write "Resolved string is " & resolved
```

Resolves the variable `<WINSYSDIR>` in the string and writes the resolved string to the log file.

8.2.5 NwSapSetupATLCommon.TextFileParser

`NwSapSetupATLCommon.TextFileParser` offers a method for reading and writing text files and searching and modifing text in text files.

Properties

Property Name	Description
`FileName( )` [page 139]	Returns the name of the parsed text file
`Parse(string filename)` [page 140]	Reads a text file from disk. Returns true if file exists and can be read or false if the file does not exist
`ReplaceLine( string LineToReplace, string Replacement)` [page 140]	Replaces all occurrences of the lines that match `LineToReplace` with the replacement string
`ReplaceLineEx(string StringToReplace,string Replacement)` [page 139]	Replaces all lines that contain the given string by the given replacement
`DoesLineExist(string Line)` [page 141]	Returns true if one or more lines in the file completely match the given value, else false
`DoesStringExist(string bstrStringToSearch)` [page 142]	Returns true if one or more lines in the file contain the given value, else false
`Save(string FileToSaveAs)` [page 143]	Saves the file with the given file name. Returns true if the file was saved successfully, else false
`AppendLine(string NewLine)` [page 143]	Appends the given string at the end of the text file
`NumLines()` [page 144]	Returns the number of lines in the file
`Line(ULong LineIndex)` [page 144]	Returns the line with the given line index
`RemoveLine(string LineToRemove)` [page 145]	Removes all lines from the text file that match the given line

NwSapSetupATLCommon.TextFileParser.FileName

Syntax

```
<objTextFileParser>.FileName()
```

Returns

Returns the name of the opened file.

Remarks

If no file was parsed `FileName` returns `NULL`.

Sample Code

```
Dim objTextFileParser
set objTextFileParser = CreateObject("NwSapSetupATLCommon.TextFileParser")
objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services")
dim filename
filename = objTextFileParser.FileName
NwEngine.Context.Log.Write filename
```

Returns the file name of a text file parser.

NwSapSetupATLCommon.TextFileParser.Parse

Reads a text file from disk

Syntax

`<objTextFileParser>.Parse(String filename)`

Arguments

`filename` required. Path to the text file to read.

Returns

Returns `Bool: true` true if file could be successfully red, `Bool: false` if not.

Sample Code

```
Dim objTextFileParser
set objTextFileParser = CreateObject("NwSapSetupATLCommon.TextFileParser")
objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services")
dim filename
filename = objTextFileParser.FileName
NwEngine.Context.Log.Write filename
```

Parse a file from disk

NwSapSetupATLCommon.TextFileParser.ReplaceLine

Replace all occurrences of the lines that match LineToReplace with the replacement string.

Syntax

`<objTextFileParser>.ReplaceLine(String oldLine, String newLine)`

Arguments

`oldLine` required. Line that match this text will be replaced.

`newLine` required. New content of the matching lines.

Remarks

The whole line is compared to the parameter oldLine. Use ReplaceLineEx if you want to match parts of the lines.

If more than one line matches the given value all lines are replaced.

Sample Code

```
Dim objTextFileParser
Set objTextFileParser = CreateObject("NwSapSetupATLCommon.TextFileParser")
If (objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services")) Then
    objTextFileParser.ReplaceLine "sappcd 3500/tcp","sappcd 3500/udp #this
line is changed"
    objTextFileParser.Save "C:\Windows\System32\drivers\etc\services"
End If
```

Replace line by its content

NwSapSetupATLCommon.TextFileParser.ReplaceLineEx

Replace all occurrences of the lines where a part of the content matches the search criteria with the replacement string.

Syntax

```
<objTextFileParser>.ReplaceLineEx(String searchString, String newLine)
```

Arguments

`searchString` required. Search criteria that contain this text will be replaced. newLine required. New content of the replaced lines.

Remarks

All lines that contain the search criteria will be replaced by the given value.

Sample Code

```
Dim objTextFileParser
Set objTextFileParser = CreateObject("NwSapSetupATLCommon.TextFileParser")
If (objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services")) Then
    objTextFileParser.ReplaceLineEx "sappcd 3500","sappcd 3500/udp #this line
is changed"
    objTextFileParser.Save "C:\Windows\System32\drivers\etc\services"
End If
```

Replace line by part of its content

NwSapSetupATLCommon.TextFileParser.DoesLineExist

Checks the existence of a line.

Syntax

<objTextFileParser>.DoesLineExist(String searchCriteria)

Arguments

`searchCriteria` required. Content of line to be verified.

Returns

Returns `Bool: True` if line could be found, else `Bool:false`.

Sample Code

```
Dim objTextFileParser
set objTextFileParser = CreateObject("NwSapSetupATLCommon.TextFileParser")
If (objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services")) Then
    If (objTextFileParser.DoesLineExist("login 513/tcp #Remote Login")) Then
           MsgBox("Line for Remote login service found")
    End If
End If
```

Check existence of a line by comparing complete content of line.

NwSapSetupATLCommon.TextFileParser.DoesStringExist

Checks whether the given string exists in the file.

Syntax

`<objTextFileParser>.DoesStringExist(String searchCriteria)`

Arguments

`searchCriteria` required. String to be searched for.

Returns

Returns `Bool: True` if string could be found, else `Bool:false`.

Sample Code

```
Dim objTextFileParser
set objTextFileParser = CreateObject("NwSapSetupATLCommon.TextFileParser")
If (objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services")) Then
    If (objTextFileParser.DoesStringExist("513/tcp")) Then
           MsgBox("Line for Remote login service found")
    End If
End If
```

Check existence of a line by a search string.

NwSapSetupATLCommon.TextFileParser.Save

Saves the data of the TextFileParser object to a file.

Syntax

```
<objTextFileParser>.Save(String filename)
```

Arguments

`filename` required: Name of the file where the TextFileParser stores its data.

Returns

Returns `Bool: true` if successfully save, else `Bool:false`.

Remarks

If the file to be saved already exists the existing file will be deleted before the new content is saved.

All data in the existing file is deleted.

Sample Code

```
If (objTextFileParser.Save("C:\Windows\System32\drivers\etc\services")) Then
    MsgBox("Services file successfully save.")
End If
```

Save buffer of TextFileParser object to disk

NwSapSetupATLCommon.TextFileParser.AppendLine

Appends a line to the text buffer of the TextFileParser object.

Syntax

```
<objTextFileParser>.AppendLine(String line)
```

Arguments

`line` required: Text to be appended to the buffer.

Remarks

AppendLine appends the text to the internal buffer of the TextFileParser.

If you want to append an existing file you have to parse it in the first step.

Sample Code

```
Dim objTextFileParser
Set objTextFileParser = CreateObject("NwSapSetupATLCommon.TextFileParser")
If (objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services")) Then
    objTextFileParser.AppendLine("sapmsXYZ 75988/tcp")
    objTextFileParser.Save "C:\Windows\System32\drivers\etc\services"
```

```
End If
```

Append a line at the end of the file.

NwSapSetupATLCommon.TextFileParser.NumLines

Returns the number of lines in the text buffer of the TextFileParser object.

Syntax

`<objTextFileParser>.NumLines()`

Returns

Returns `Long`: number of lines in the buffer.

Remarks

If a file was parsed NumLines returns the number of lines red from the file. If no file was parsed NumLines return 0

Sample Code

```
Dim objTextFileParser
Set objTextFileParser = CreateObject("NwSapSetupATLCommon.TextFileParser")
If (objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services")) Then
    msgbox("Services file has " + objTextFileParser.NumLines + " lines")
End If
```

Check number of lines in buffer and save to disk.

NwSapSetupATLCommon.TextFileParser.Line

Returns the line indicated by the given index.

Syntax

`<objTextFileParser>.Line(Long index)`

Returns

Return `String`: content of the line indicated by the given index.

Arguments

`index` required: Number of the line in the buffer.

Remarks

The first line of the buffer is indeicat by 1. The function fails if a line outside the buffer is accessed. Use NumLines in the first step to get the number of lines in the buffer.

Sample Code

```
Dim objTextFileParser, line
Set objTextFileParser = CreateObject("NwSapSetupATLCommon.TextFileParser")
If (objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services") And
(objTextFileParser.NumLines > 0)) Then
    line = objTextFileParser.Line(1)
    NwEngine.Context.Log.Write "Line 1: " & line
End If
```

Parse a file and return first line if it exists

NwSapSetupATLCommon.TextFileParser.RemoveLine

Removes a line from the text buffer of the TextFileParser object.

Syntax

`<objTextFileParser>.RemoveLine(String line)`

Arguments

`line` required: Text to be removed from the buffer.

Remarks

RemoveLine removes text from the internal buffer of the TextFileParser.

Sample Code

```
Dim objTextFileParser
Set objTextFileParser = CreateObject("NwSapSetupATLCommon.TextFileParser")
If (objTextFileParser.Parse("C:\Windows\System32\drivers\etc\services") And
(objTextFileParser.NumLines > 0)) Then
    If objTextFileParser.DoesLineExist("sapgw99  3399/tcp") Then
        objTextFileParser.RemoveLine("sapgw99  3399/tcp")
        objTextFileParser.Save "C:\Windows\System32\drivers\etc\services"
    End If
End If
```

Remove first line from the buffer.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

© 2026 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.