Admin

idgard Sync

Table of Contents

Admin

Welcome to *Admin* of **idgard Sync**.

This page is additionally available as 📰 **PDF** for **Download** 📰!

The idgard Sync is a Windows app that brings your idgard workspace directly into your daily desktop workflow.

> ⓘ **NOTE**
> With idgard Sync, your files are available in Windows Explorer just like a familiar cloud drive, while keeping the security standards of idgard.

Installer

System Requirements

The following requirements must be met to install and run idgard Sync.

Operating System

Requirement	Minimum
OS	Windows 11 (64-bit)

> ⊗ **IMPORTANT**
>
> **Windows 10** is **not** officially supported. The application may run on Windows 10 22H2 (Build 19045) or later — the underlying Windows SDK targets Build 19041 (20H2), which covers the same API surface shared by 20H2, 21H1, 21H2, and 22H2. However, feature gaps and stability issues are possible and will not be addressed.

Runtime Dependencies

The installer automatically downloads and installs any missing runtime components.

Component	Version
.NET Desktop Runtime	8.0.26 (x64)
Windows App Runtime	1.8.7 (x64)

> ⓘ **NOTE**
>
> Both runtime components are bundled with the exe-installer and will be set up automatically if not already present on the system. They are not included in the MSI setup and must be installed separately beforehand.

Update Policy

This document describes how idgard Sync tracks and adopts updates to its core dependencies.

Component	Strategy	Lag after support/release end
Windows OS (feature update)	Raise minimum after MS support ends	3 – 6 months
Windows App SDK (major version)	Stay on latest major ≥ 6 months old	≥ 6 months
.NET Desktop Runtime (major/LTS)	Adopt LTS versions only	6 – 12 months after GA
.NET Desktop Runtime (patch)	Always use latest patch of current LTS	Immediate

Windows Operating System

We follow Microsoft's servicing timeline for Windows feature updates.

Rule: We raise the minimum supported Windows version **3 to 6 months after Microsoft ends mainstream support** for the version currently listed as the minimum. The exact timing within that window depends on release readiness and testing.

Resource	Link
Windows 10 release information	learn.microsoft.com – Windows 10
Windows 11 release information	learn.microsoft.com – Windows 11

Windows App SDK

Rule: We target the **latest major version that is at least 6 months old** at the time of a release. We do not adopt a brand-new major version immediately; we wait until it has been available for at least 6 months before making it the required minimum. Within the chosen major version we always ship the latest available release.

Resource	Link
Windows App SDK downloads & release notes	learn.microsoft.com – Windows App SDK

.NET Desktop Runtime

Rule: We target **Long-Term Support (LTS) releases only** (e.g., .NET 8, .NET 10). A new LTS version is adopted **6 to 12 months after its GA release**, once the ecosystem has stabilized. Within the adopted major version we always ship the **latest minor or patch release** (e.g., 8.0.26 today).

Non-LTS / Standard-Term Support (STS) releases are not used as the required runtime.

Resource	Link
.NET download page & release schedule	dotnet.microsoft.com

Deployment Guide

This guide explains common ways to deploy an MSI package, from interactive installation to fully automated rollout.

> ⓘ **NOTE**
> Use **Full UI** for manual user installs.
> Use **Silent** (`msiexec`) for enterprise automation.
> Use **Custom Properties** for install-time options (e.g. AppUpdate yes/no).
> Use **Config Files** for post-install environment customization.

Best practices

- Keep deployment steps predictable and repeatable
- Use logging for unattended installs
- Validate the installation result after rollout

Full UI Installation (Next → Next → Finish)

Use this when a user installs manually on a workstation (user needs admin-rights).

Steps

1. Double-click the `.msi` file.
2. Follow the installer wizard.
3. Click **Next** through the dialogs.
4. Click **Finish** at the end.

> ⓘ **NOTE**
> **Best for**
>
> - manual installs
> - Small-scale business
> - IT-assisted guided setup

Silent Installation (`msiexec` with no UI)

Use this for unattended deployment via scripts, Intune, SCCM, GPO, or RMM tools.

```
msiexec /i "setup.msi" /qn /norestart
```

Useful options

- `/i` = install package
- `/qn` = no UI (silent)
- `/norestart` = do not restart automatically
- `/l*v "C:\Temp\install.log"` = verbose logging

Example with logging:

```
msiexec /i "setup.msi" /qn /norestart /l*v "C:\Temp\install.log"
```

> **NOTE**
>
> **Best for**
>
> - installs requiring detailed logging
> - Mid-scale business
> - IT-centralized setup

Install with **Custom Properties**

Use MSI public properties (must be in **UPPERCASE**) to control installer behavior.

See Installer Custom Variables (MSI Public Properties) to see all custom properties of **idgard Sync**.

Example: Silent install with boolean `MY_BOOLEAN_PROPERTY=1` - 1 to enabled a property (yes, true)

```
msiexec /i "setup.msi" /qn MY_BOOLEAN_PROPERTY=1
```

Example: Silent install with boolean `MY_BOOLEAN_PROPERTY=0` - 0 to disable a property (no, false)

```
msiexec /i "setup.msi" /qn MY_BOOLEAN_PROPERTY=0
```

Example: Silent install with string `MY_STRING_PROPERTY` - wrap the entire `NAME=Value` pair in double quotes `"NAME=My Value"` when the value contains spaces

```
msiexec /i "setup.msi" /qn "MY_STRING_PROPERTY=Lorem Ipsum"
```

Common patterns

- `MY_BOOLEAN_PROPERTY=1` or `MY_BOOLEAN_PROPERTY=0` (boolean)
- `"MY_STRING_PROPERTY=Lorem Ipsum"` (string with spaces — quote the whole pair)

Post-Deployment Configuration via Config Files

After MSI deployment, apply environment-specific settings using config files.

We have a special config folder `C:\Users\<USERNAME>\AppData\Local\<PRODUCTNAME>\config`.

Typical flow

1. Deploy MSI package.
2. Copy configuration file(s) (for example: .json, .config, .xml, .ini) to **config** folder.
3. Set **secure values** (certs, tokens, endpoints) via **secret management/tooling**.
4. Restart app.

Example script flow

```
msiexec /i "setup.msi" /qn /norestart
Copy-Item "C:\IT\configs\my-config.json" "C:\Users\<USERNAME>\AppData\Local\
<PRODUCTNAME>\configs\my-config.json" -Force

# Restart Application, over the start menu
```

Installer Custom Variables (MSI Public Properties)

The following **custom properties** (in UPPERCASE) are public MSI properties and can be set at install time via the `msiexec` command line tool.

See also the Deployment Guide for some generic examples.

> **NOTE**
> MSI public properties require a `NAME=VALUE` pair.
>
> **Examples:**
> `MY_BOOLEAN_PROPERTY=1` or `MY_BOOLEAN_PROPERTY=0` (boolean)
> `"MY_STRING_PROPERTY=Lorem Ipsum"` (string with spaces — quote the whole pair)

> **WARNING**
> The name alone is not valid, and the property will be **ignored**.
> Property names (in uppercase) must match what the MSI supports, if a property is not known, it will be ignored **without** a warning or error.

Usage Examples for Custom (overwritable) Properties

Custom properties control both the **installer behavior** and the **runtime behavior** of the installed application. Properties that affect runtime are persisted to the Windows Registry under `HKLM\SOFTWARE\`**idgard Sync** during installation. The application reads these values at startup or on demand.

In the examples below we **omit** optional `msiexec` parameters, to keep the focus on the usecase of the custom property.

Override the **Application Update Functionallity**, set it to `1` to `disable the auto update` (default value is `0`, `update is active`):

```
msiexec /i "setup.msi" /qn DISABLE_APP_UPDATE="1"
```

PROPERTY	RegistryKey	Values	Use Case
DISABLE_APP_UPDATE	DisableAppUpdate	**0**, 1	Control Application Update

> **NOTE**
> Only properties with an all-uppercase `PROPERTY_NAME` in the WiX source are public MSI properties. Properties marked `Secure="yes"` are populated internally and cannot be overridden on the command line.

Standard Properties

> **WARNING**
> Remember **Standard Properties** are exposed but should **not** modified by user.

These **ARP properties** control how the product appears in **Add/Remove Programs** (Programs and Features).

Property	Description
`ARPCONTACT`	Support contact shown in Add/Remove Programs.
`ARPHELPLINK`	Help URL shown in Add/Remove Programs.
`ARPHELPTELEPHONE`	Support phone number shown in Add/Remove Programs.
`ARPPRODUCTICON`	Icon shown in Add/Remove Programs.

Internal Properties

Property	Description
`EXENAME`	Name of the main executable.
`TENANT_PRODUCT_NAME`	Display name of the tenant product.

Secure / **Read-Only** Properties

All secure properties are set internally (`Secure="yes"`) and **cannot** be set from the command line!

Property	Description
`PREVIOUSVERSIONSINSTALLED`	Set by the upgrade table.

Config Files

We have a special config folder `C:\Users\<USERNAME>\AppData\Local\<PRODUCTNAME>\config`, copy configuration file(s) (for example: .json, .config, .xml, .ini) to config folder restart app.

Config files are part of the Deployment Guide.

Common usage of config files:

1. In Windows Explorer navigate to `%LocalAppData%/APP_NAME/`.
2. Create a folder `config` if it does not exist.
3. Create a new json file with the name of the configuration, e.g. `my-config.json`.

`APP_NAME` is idgard Sync.

Behavior (`_behavior` property)

All JSON config files require a `_behavior` setting that controls how the configuration is applied on startup. Not every setting supports all `_behavior` variants.

If `_behavior` is not set or the value is not recognized, `once` is used.

Value	Description
`once` *(default)*	Reads the config file, imports the values, then deletes the file. The user can change the values later in the app settings.
`always`	Reads the config file, imports the values, and keeps the file. The values will always overwrite user settings on every startup.
`OnceIfNotSet`	Like `once`, but only imports a value if no existing app setting is found for it.

Web Proxy

Configure a web proxy via config-file.

- Config **filename**: `proxy.json`
- Supported **_behavior**: `once`, `always` and `OnceIfNotSet`.

Example of `proxy.json`.

```
{
    "_behavior": "once",
    "ProxyAddress": "my.proxy-server.tld",
    "ProxyPort": 8080
}
```

Configure `ProxyAddress` (can be an IP or an URL) and `ProxyPort` to your needs.

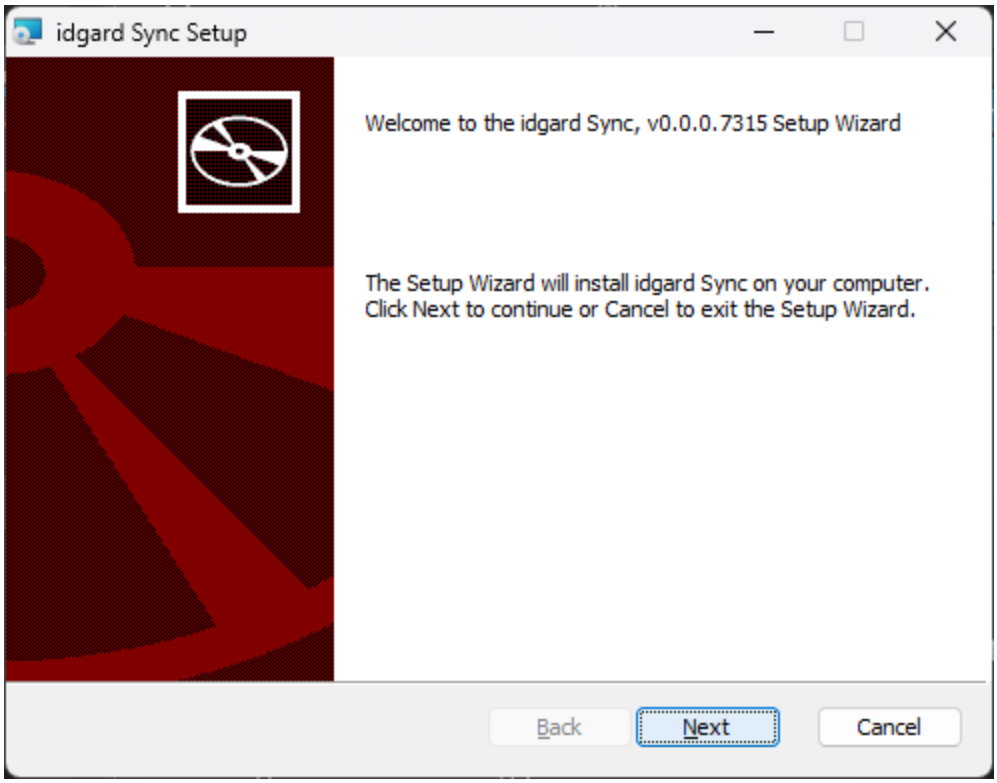
The usage of the proxy will also be seen in the log-files

```
[YYYY-mm-dd HH:MMM:ss.fff INF] Parsing the proxy config file
'C:\Users\USER_NAME\AppData\Local\APP_NAME\config\proxy.json' was successful. Address:
```

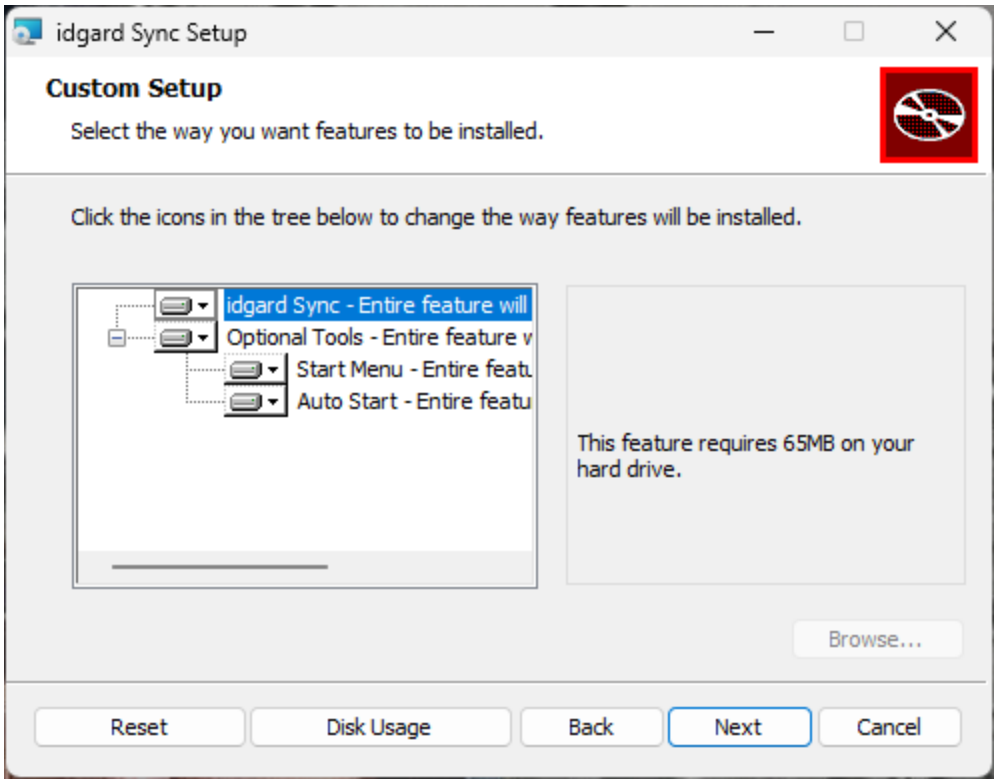
```
my.proxy-server.tld, Port: 8080
```

MSI-Setup Screenshots

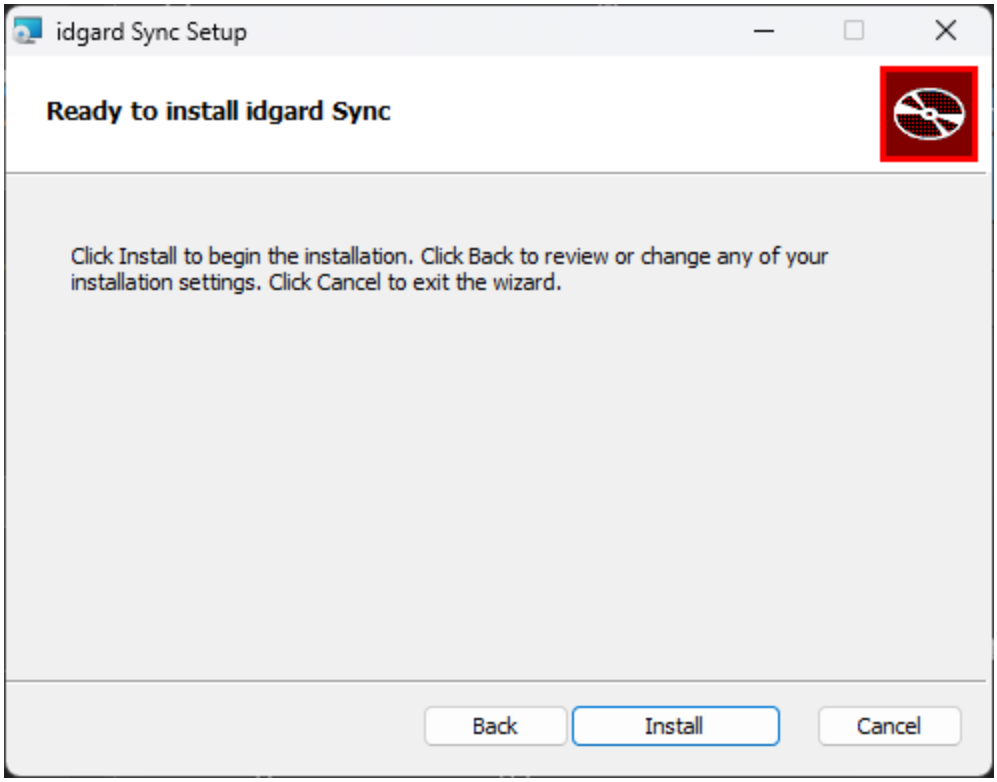
We are using an next-next-next-finish setup approach



Welcome Dialog

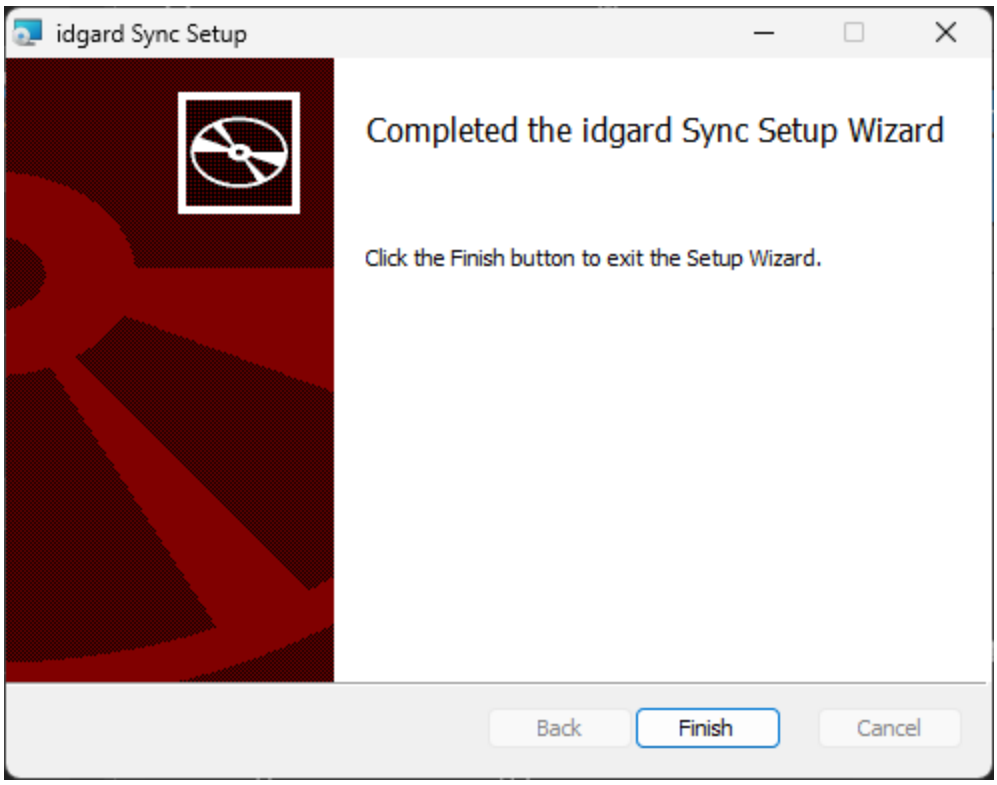


Feature Tree



Confirm Dialog

> UAC – User Access Control to ask for admin permission.



Finish Dialog