Updating and upgrading Webdesk 4.x

This page describes how an update or upgrade of **Webdesk 4.x** works and how to carry it out safely. It explains what Webdesk does automatically when a new program version starts (database schema, module upgrade scripts, registration) and which settings play a role. The audience is system administrators and consultants.

The page applies to Webdesk 4.x operated as a WAR file in a Tomcat or as a container image. The update with the Windows installer (setup-...exe) is described on the page Webdesk Update[1]; this page complements it with the background and the procedures for WAR and container operation. The initial installation is described under Webdesk 4[2].

Supported Java versions per Webdesk version

The table shows from which Webdesk version the minimum required Java version changed and which additional Java version was already supported. The boundaries were checked against the compiler level of the build of the respective version (source/target or release in the Maven configuration) and, where available, are backed by the release notes. Version numbers are not strictly chronological: hotfix versions of older lines also appear after newer major versions and keep the Java requirement of their line.

Webdesk version	Minimum Java	Additionally supported
4.0.0 to 4.1.1	Java 7 (derived from the build, not explicitly named in the release notes)	none
4.1.2 to 4.19.x	Java 8	none (Java 11 is not yet supported)
4.20.0 to 4.57.x	Java 8 (from 4.56.0 at least update 163, see New features in 4.56[3])	Java 11 from 4.20.0 (release notes 4.20.0[4]: Java 11 requires at least Tomcat 9)
4.58.0 to 4.61.x	Java 11 (release notes 4.58.0[5]: "Webdesk is now based on Java 11")	none
4.62.0 to 4.63.x	Java 11	Java 17 from 4.62.0 (release notes 4.62.0[6]: Nashorn is embedded to enable operation with Java 17); additional JVM parameters required, see Webdesk with Java 17[7]
4.64.0 and later	Java 17 (Breaking Changes 4.64[8])	none; Java 11 is not sufficient

Java 21 or later is not mentioned in the release notes or in the build; use Java 17 for 4.64 and later. When operated as a container image, Java and Tomcat are already included in the image.

When an update crosses a Java boundary:

- **Update to 4.58 or later:** provide Java 11 and Tomcat 9 on the Webdesk server (Java 11 requires at least Tomcat 9).
- **Update to 4.64 or later:** install Java 17 on the Webdesk server and for the IF6020 TA connector. Java 11 is not sufficient. Check which Java version the Windows service or start script actually uses.
- **Under Java 17:** set the startup parameters --add-opens (setenv.sh or wrapper.conf) and remove the parameter -Dnashorn.args=--no-deprecation-warning (Webdesk with Java 17[9], New features 4.64[10]).

- **Update to 4.56 or later:** use at least Java 8 update 163 and create a full backup of the databases beforehand (New features in 4.56[11]).
- Test customer-specific Java jobs and integrations after the switch.

Prerequisites

- **Java 17:** Webdesk 4.64 and the IF6020 TA connector run on Java 17; Java 11 is not sufficient for this version (see Breaking Changes 4.64[12] and the table above). The supplied container image is based on a JDK 17. If several Java installations exist, check which Java version the service or start script actually uses.
- **Tomcat 9:** The container image (webdesk-webapp/Dockerfile) is based on a Tomcat 9 image with JDK 17. The Windows installer also ships a Tomcat 9.
- **JVM parameters --add-opens:** Depending on the way Webdesk is operated, the following parameters are required when the JVM starts (list according to New Features 4.64[13]):

```
--add-opens java.base/jdk.internal.loader=ALL-UNNAMED
--add-opens java.base/jdk.internal.module=ALL-UNNAMED
--add-opens java.base/jdk.internal.reflect=ALL-UNNAMED
--add-opens java.base/java.lang.module=ALL-UNNAMED
--add-opens jdk.crypto.ec/sun.security.ec.point=ALL-UNNAMED
--add-opens jdk.dynalink/jdk.dynalink.linker.support=ALL-UNNAMED
--add-opens java.base/sun.util.calendar=ALL-UNNAMED
--add-opens java.base/java.util.concurrent.locks=ALL-UNNAMED
--add-opens java.base/java.lang=ALL-UNNAMED
--add-opens java.base/java.lang.invoke=ALL-UNNAMED
--add-opens java.base/java.lang.reflect=ALL-UNNAMED
--add-opens java.base/java.io=ALL-UNNAMED
--add-opens java.base/java.util=ALL-UNNAMED
--add-opens java.rmi/sun.rmi.transport=ALL-UNNAMED
--add-opens java.base/java.net=ALL-UNNAMED
```

  Set the parameters where the JVM is started, e.g. through the environment variable JDK_JAVA_OPTIONS (this is how the catalina.bat supplied with the Windows installer sets some of them) or through the Tomcat start options. In a container they must be passed to the container's JVM; the Dockerfile of the web application does not set them itself.
- **Source version:** According to Webdesk Update[14], Webdesk versions older than 3.8 first require an update to the latest 3.8.x.
- **Notes on the target version:** Before every update, read the "New Features" and "Breaking Changes" pages of the target version, e.g. Breaking Changes 4.64[15]. When updating to Webdesk 4.56 or higher, stored credentials are re-encrypted, see New Features in 4.56[16].

What happens when a new version starts

At its core, an update consists of replacing the program files and restarting. Webdesk performs all further steps itself during startup, in this order:

1. **Hibernate schema update:** When the Hibernate session factory is built, missing tables and columns of the Webdesk database are added automatically (property webdesk-sessionFactory.schemaUpdate, default: enabled). The schema update does not perform renames, deletions or data migrations; that is what the module upgrade scripts are for.
2. **Index manager:** Compares the existing database indexes with the expected ones and creates missing ones (PoStartup.executeIndexManager, default: enabled; errors here only cause a warning in the log).
3. **Module upgrade scripts:** For each module, the upgrade steps of the new version are executed (see below). This step runs on every startup and cannot be switched off.
4. **Checks:** Database consistency checks (for MySQL, among others, the character set) and a timeline check of the persons (can be switched off with webdesk.disableTimeLineChecks).

5. **Licence** and **system parameters:** The licence definitions are loaded. New system parameters (bean properties) are written to the database, existing values are kept, and all values are applied.
6. **Registration:** Actions, configurations, text modules, jobs etc. of the modules are transferred to the database. Whether and how this happens is controlled by webdesk.runRegistration (see below).
7. **Full-text index:** The search index is refreshed (webdesk.luceneIndexing, default: enabled).

Relevant settings in webdesk.properties

Property	Description	Possible value(s)
webdesk.runRegistration	Controls whether and how the registration runs at startup (values see below). The value 1 is automatically reset to 0 after the registration. Can also be set as the environment variable WD_WEBDESK_RUNREGISTRATION.	0, 1, 2, 3, 4 (default: 1 if not set or invalid; the supplied template sets 1)
webdesk.runRegistrationInThread	If true, the registration runs in a background thread ("Webdesk Artifact Registration") while Webdesk is already starting. If false, it runs in the foreground and startup finishes only afterwards.	true, false (default: true)
webdesk-sessionFactory.schemaUpdate	If true, Hibernate adds missing tables and columns at startup. If false, the schema is not adjusted; Webdesk then assumes that the database already matches the program version. There is no WD_ environment variable for this setting.	true, false (default: true)
PoStartup.executeIndexManager	If true, the database indexes are checked at startup and missing ones are created. This can take long on large databases. The template webdesk.properties.template uses the older name webdesk-SessionFactory.executeIndexManager and sets it to false.	true, false (default: true)

Module upgrade scripts

How the version decides what is executed

Each module (po, wf, ta, gw, ...) has its own version number, a consecutive integer that is independent of the Webdesk release number. The current state per module is stored in the database and shown in the action "Modules" in the column "Version". The program brings the target version for each module. At startup:

- If the stored version equals the target version, nothing happens (log: Module ... is up-to-date).
- If the stored version is lower, the files in the module's folder .../update/vNN/ are executed for every version in between (NN is the respective version number). First the prepare.xml files of all pending

versions run, then the upgrade.xml files version by version. After each successfully completed upgrade.xml the new version number is saved in the database.

- A newly added module (e.g. after a licence extension) is created directly with the target version; no upgrade steps are executed.
- Modules that no longer exist or are no longer licensed in the new version are set to "detached"; they are reactivated as soon as they are available again.
- The modules are always processed in the same order, which is determined by their dependencies, and each module is completed entirely (prepare and upgrade) before the next one begins.

If the stored version is higher than the target version (older program version against a newer database), no script is executed; the stored version number is merely set to the lower target version. See the section "Rolling back".

Structure of an upgrade file

The scripts reside in the module under /at/workflow/webdesk/<module>/<implFolder>/update/vNN/ and are named prepare.xml or upgrade.xml. The root element is arbitrary (usually <upgrade>); its child elements are processed in order:

Element	Description	Attributes
<sql>	Executes the SQL statement contained in the element.	dataSource (webdesk or shark), failOnError, applies and excludes (database type, comma-separated: mysql, mssql, oracle, db2, hsql; for applies also *)
<js>	Executes the JavaScript file whose name is given in the element. The file resides in the same vNN folder.	failOnError
<java>	Executes the Java class named in the element (fully qualified name). The class must derive from PoAbstractUpgradeScript; it runs in a transaction unless the class specifies otherwise.	failOnError
<rename>	Renames a system parameter (bean property).	type (currently only beanproperty), old, new, new-module, failOnError
<condition>	Encloses any other elements, which are executed only if all given conditions are met. Without attributes the condition is always met.	dataSource, tablesExist, columnsExist (table.column), columnsNotExist

Important for error handling: failOnError is **false** by default. A failed step is then only recorded in the log, and the upgrade continues and counts as successful.

Finding results and errors

There is no separate mask with a log of the upgrade steps. The results are in the Webdesk log (file webdesk.log or error.log in the folder webdesk-logs below the Cocoon work directory, as well as on the console or in the container log). Relevant messages:

Log message	Meaning	Action required
Starting to upgrade module ... from vX to vY and Finished upgrading module ...	Start and end of the upgrade of a module.	None.
Executing .../vNN/upgrade.xml (or prepare.xml)	Which file is currently being executed.	None.
Upgrade script couldn't be executed normally! followed by Failed sql command is: ...	A step with failOnError false failed; the upgrade continued.	Check every message. Some steps are deliberately designed so that they may fail on certain databases; if in doubt, ask support.
Couldn't upgrade module ... vX to vY	A step with failOnError true or an unreadable upgrade file aborted the upgrade.	See "If an upgrade step fails".

Registration

What the registration does

The program files of a new version bring actions (act-descr.xml), configurations, text modules (i18n.xml), jobs, connectors, key values, help texts and menu templates. The registration transfers them to the database. Without registration, the database contents stay at the level of the previous version: new actions, menu entries and captions are then missing.

The registration runs in two phases: phase 1 ("ESSENTIALS") registers everything except the text modules, phase 2 ("LABELS (i18n)") the text modules. It is compute-intensive and can take several minutes.

Values of webdesk.runRegistration

Value	Behavior at startup	Typical use
0	No registration.	Normal operation without an update.
1	Full registration (both phases) **ignoring the INI file cache**; afterwards the value in webdesk.properties is automatically set to 0.	Default for an update. Runs only once.
2	Full registration on every startup (first everything except text modules, then the text modules). The INI file cache is used.	Test and development systems.
3	Registration without text modules on every startup.	When text modules are not to be touched.
4	Like 2, but after completion the Webdesk JVM exits with exit code 0. According to the source code intended only for automated deployments and updates with Kubernetes (not documented in the template).	One-off update container, see section Container.

Notes:

- If the database is still empty (new installation), a full registration is always run in the foreground, regardless of the setting.
- The automatic reset from 1 to 0 writes to the file webdesk.properties. If this file does not exist (e.g. in a container), nothing is reset and the value 1 applies on every startup. If the value is set through the environment variable WD_WEBDESK_RUNREGISTRATION, the variable remains authoritative: remove it after the update or set it to 0, otherwise a full registration runs on every startup.
- By default the registration runs in the background and the application is reachable in the meantime. Admit users only when REGISTRATION HAS FINISHED (or REGISTRATION All Phases finished!) appears in the log.
- The **INI file cache** (files actions.ini and textmodules.ini in the subfolder inifiles of the cache directory) remembers for every act-descr.xml and i18n.xml when the file was last changed. A file that is not newer than the remembered state is skipped. If the cache is ignored, all files are read again. The cache is bound to its database: if the database connection changes, it is discarded.
- Customer-specific changes are protected: shipped configurations and text modules are overwritten by the registration only if the flag "Update on version change" (allowUpdateOnVersionChange) is set on them. Newly created configurations do not get it (so they are not overwritten afterwards). The flag of all text modules can be set or removed in bulk with the actions "Unlock all Textmodules in Database" and "Lock all Textmodules in Database" in "Extended Tasks" (see Extended Tasks[17]); "Unlock all Textmodules in Database" allows the registration to overwrite all shipped text modules with the original values.

Triggering the registration manually

In the action "Extended Tasks" (Extended Tasks[18]), the registration can be started during operation, e.g. if webdesk.runRegistration was set to 0 or individual modules failed:

- "Registration mode": *complete* (everything), *Everything except Textmodules*, *Actions and Flowscripts* (without jobs and job configurations) or *Only text modules*.
- "Ignore ini-file cache": selectable only for *complete*; corresponds to the behavior with value 1.
- "Register only this module:": registers only the selected module.
- Button "Run Registration".
- In addition there are "Clear Ini-File Caches" (the next registration then reads all files again) and "Start Update process of modules" (executes the module upgrades as at startup, without a restart).

Warning: The message "Registration successfully finished!" also appears when the registration of individual modules has failed, because errors are only written to the log per module. Always verify success in the log. Do not run the registration during business hours.

Checklist before the update

- Release notes ("New Features", "Breaking Changes") of the target version read; Java version according to the table above and the JVM parameters checked.
- Maintenance window agreed, users informed. Check in "Active Jobs" (Active Jobs[19]) that no jobs are running any more.
- **Backup** (details on the page "Backup, Restore und Verschlüsselungsschlüssel"):
  - Full backup of the databases **webdesk** and **shark**.
  - The files webdesk.properties and enc.key (located on the classpath of the web application, usually WEB-INF/classes) and, if applicable, tacon.properties of the TA connector. Without the encryption key, stored credentials can no longer be decrypted.
- For systems with customer-specific adaptations: test the update beforehand on a copy of the databases.

- Save the current log files: according to log4j2.xml, log files are not appended to but started anew when the new version starts.
- For operation with several nodes (cluster): decide on the procedure for the first node (see below).

Update with a classic Tomcat (WAR)

1. Complete all preparations of the checklist, log users off.
2. Stop Tomcat or the Webdesk service.
3. Perform the backup (databases, webdesk.properties, enc.key, logs).
4. Set webdesk.runRegistration = 1 in webdesk.properties. Reason: after the first start the value is 0, so the registration would otherwise be skipped and new actions, menu entries and captions would be missing. The module upgrade scripts run independently of this setting, always.
5. Remove the old web application (unpacked folder) and deploy or unpack the new WAR. The saved webdesk.properties and enc.key must afterwards be back on the classpath (WEB-INF/classes) of the new web application, as must any other customer-specific files stored there.
6. Start Tomcat and watch the Webdesk log until startup has finished and the registration is complete (messages WebdeskStartup PoStartupImpl.start() finished after ... and REGISTRATION HAS FINISHED AFTER ...; the latter appears once per phase).
7. Carry out the post-update verification (see below).

Update with a container image

1. The new image does not contain a webdesk.properties. Configuration and key (webdesk.properties, enc.key) must be provided as before. Alternatively, the following settings can be set through environment variables of the form WD_<PROPERTY> (dots become underscores, upper case; e.g. webdesk.connection.url becomes WD_WEBDESK_CONNECTION_URL): database connections (webdesk.connection.*, shark.connection.*, webdesk.dialect), webdesk.ta.if6020impl.rmi, cluster settings, webdesk.runRegistration, webdesk.disableTimeLineChecks and a few more.
2. Back up the databases, stop the existing containers.
3. Decide on the registration, either
   - (recommended for automated procedures) start a one-off update container with webdesk.runRegistration = 4. It performs the schema update, module upgrades and the full registration, and then terminates itself. Only then start the regular containers with value 0; or
   - start the regular containers with value 1.
4. With several nodes using the same database, start only one node first and wait for its complete startup before starting further ones. Reason: the upgrade steps are executed when each node starts.
5. Remove the environment variable WD_WEBDESK_RUNREGISTRATION after the update or set it to 0 (see above).
6. Carry out the post-update verification.

Note: The web application image contains Tomcat 9 and the session storage through Redis (context.xml). The Redis service must be reachable for the new container.

Verification after the update

1. **Log:** There are no messages Couldn't upgrade module, Registration of ... module failed or Action Registration failed!. Messages Upgrade script couldn't be executed normally! have been reviewed.
2. **Login:** Logging in as administrator is possible; the action "Versions-Popup" shows version, revision and build time of the new program version.

3. **Modules:** In the action "Modules" all expected modules are present and the column "detached" is false for the licensed modules. After another restart, the log shows is up-to-date for all modules, i.e. all upgrade steps are complete.
4. **Jobs:** In "Active Jobs" the expected jobs are scheduled and running. If not, run the action "Reinitialization of JobService" in "Extended Tasks".
5. **Caches:** The registration clears the caches of actions and access rights itself. If permissions still do not take effect, run "Clear Permission Cache" in "Extended Tasks"; for changed flowscripts "Refresh all flowscripts".
6. **Registration:** webdesk.runRegistration is back at 0 (do not leave operation permanently at 2, 3 or 4).
7. **Functional spot checks:** Test central processes (time recording, workflows, reports, interfaces and customer-specific scripts). For IF6020, check the TA connector and tacon.properties according to the notes of the target version.

If an upgrade step fails

What the code does in case of an error:

- If an upgrade.xml or prepare.xml aborts (message Couldn't upgrade module ...), Webdesk does not start. The remaining modules are then not processed.
- The version number of the affected module stays at the last fully completed version. At the next start Webdesk resumes at the failed version. In doing so, the steps of that version that were already executed and all prepare.xml files of the pending versions run again.
- There is no automatic rollback of the steps already executed.

Procedure:

1. Save the log files (they are overwritten at the next start) and look for the cause in the log: message Couldn't upgrade module ... with the associated error cause.
2. Fix the cause (e.g. missing database permissions for DDL statements) and restart Webdesk. Alternatively, run the action "Start Update process of modules" in "Extended Tasks"; errors are displayed in the mask there.
3. If the cause cannot be fixed, restore the backup (see next section) and contact support. Do not rework the database manually as long as it is unclear which steps have already run.

Rolling back

Warning: Webdesk 4 does not execute downgrade scripts. The readme.txt in the module's update folder does mention an optional downgrade.xml, but no such file is executed anywhere in the program (and none exists in the repository). If an older program version is started against an already updated database, no steps are reversed and the Hibernate schema update does not remove any columns; Webdesk merely sets the stored module versions to the lower values. The way back is therefore to restore the backup (databases **webdesk** and **shark** as well as webdesk.properties and enc.key) together with the old program version.

Registration failed

Errors during the registration of a module are reported in the log (Registration of ... module failed), but Webdesk keeps running. After fixing the cause, trigger the registration again in "Extended Tasks", if necessary restricted to the affected module and with "Ignore ini-file cache".

1. /daisy/personalwolke-admin/7572-dsy.html?language=4
2. /daisy/personalwolke-admin/7506-dsy.html?language=4
3. /daisy/personalwolke-admin/10823-dsy.html?language=4
4. /daisy/personalwolke-admin/8858-dsy.html?language=4
5. /daisy/personalwolke-admin/10971-dsy.html?language=4
6. /daisy/personalwolke-admin/11333-dsy.html?language=4

7. /daisy/personalwolke-admin/11359-dsy.html?language=4
8. /daisy/personalwolke-admin/11499-dsy.html?language=4
9. /daisy/personalwolke-admin/11359-dsy.html?language=4
10. /daisy/personalwolke-admin/11418-dsy.html?language=4
11. /daisy/personalwolke-admin/10823-dsy.html?language=4
12. /daisy/personalwolke-admin/11499-dsy.html?language=4
13. /daisy/personalwolke-admin/11418-dsy.html?language=4
14. /daisy/personalwolke-admin/7572-dsy.html?language=4
15. /daisy/personalwolke-admin/11499-dsy.html?language=4
16. /daisy/personalwolke-admin/10823-dsy.html?language=4
17. /daisy/personalwolke-admin/463-dsy.html?language=4
18. /daisy/personalwolke-admin/463-dsy.html?language=4
19. /daisy/personalwolke-admin/1146-dsy.html?language=4