Updating Print Director

Print Director is updated by running **PDUpdate**, a wizard that is downloaded from Blue Swift for each release and run once, on a computer that can reach the Print Director database. PDUpdate updates the central database to the new version and then loads the update packages for the other Print Director components (Resource Manager, PDServer, PDAgent, PDReports and DBTools) into the database. The services and applications pick those packages up and update themselves from there.

PDUpdate must be run as an administrator. It reads the Print Director database connection from the computer it runs on, so run it on the Print Director server, or on a computer where Resource Manager is installed and connected to the site's database.

Before you start

1. **Take a full backup of the database.** The wizard offers to do this for you on the checks page (the Backup database row). If you back up another way, make sure the backup completed before you click Next.
2. **Allow for temporary database unavailability during a purge.** You do not need to stop Print Director services. PDAgents and PDServer use offline processing where enabled and reconnect automatically when the database is available again. Other applications connected directly to the database will be disconnected.
3. **Allow time.** A normal update takes a few minutes. The 2.5.15 update rebuilds the print history into a new table and can take from a few minutes on a small site to about an hour on a database with tens of millions of print records. The wizard shows its progress the whole time (see The execution page below).

The checks page

The first page of the wizard runs a set of checks against the site and shows one row per check. Each row has a busy indicator while the check runs, then an icon and a text. Green means the check passed, amber means the update can run but you are asked to confirm when you click Next, and red means the update cannot run until the problem is fixed.

Next is enabled only when DB connection, SQL version, Database size and PD licence are all green or amber, and no previous purge has left the database in an abnormal state (see Purge history below). There is no way to override a red row.

The rows, in the order they appear:

DB connection

Confirms the wizard can reach the Print Director database. Green reads `DB connection: Online. Server address: <server>. DB: <database>. DB version: <a.b.c.d>`, where the version is the version stamped in the database before the update. Red reads `DB connection: Cannot connect to DB` and shows an **Update DB settings** button that opens the connection settings dialog (server, database, authentication, encryption). Every other check waits for this one. Until the database answers, SQL version, Database size and Purge history read `Waiting for DB connection ...`, PD licence reads `PD licence: Cannot validate licence without a DB connection.`, PDAgent .Net versions reads `PDAgent .Net versions: Waiting for valid licence.` and the Backup database row reads `Database not backed up`.

SQL version

Reads the SQL Server version. The minimum is SQL Server 2012.

- Green: `SQL version: Supported. <product version>.`
- Amber: `Couldn't get SQL version. Min version is 2012. Current version is <version text>. If you continue, the update may fail.` The update is allowed.
- Red: `Min version for MS SQL is 2012. Current version is <version text>. Please upgrade or use another SQL instance.`

Database size

New in 2.5.15. Reads the SQL Server edition, how much of the data file is used, the free space on the disk volume that holds the data file, and the size of the Printout and DeletedPrintout tables, then estimates how much the data files will grow while the update runs and decides whether the update can fit.

The estimate. While the database is below version 2.5.15, or its 2.5.15 print history rebuild has not completed, the update is estimated to need **4.5 times the size of the Printout table plus the DeletedPrintout table** in extra data-file space. That is the measured peak of the 2.5.15 history rebuild (4.1 times, on a customer database of 24 million print records) with a margin: the staging tables, the new JobSubscription rows, a second copy of that table while its primary key is rebuilt, and a copy of Printout during its own rebuild all exist at the same time before anything is dropped. The estimate covers the data files only, not the transaction log. The space is needed only while the update runs; most of it is released afterwards, though the data files stay at their grown size unless they are shrunk. Once a database is at 2.5.15 with the rebuild complete, the estimate is zero.

The rules, applied in this order:

1. **SQL Server Express, over the 10 GB limit (red, no override).** On SQL Server Express, if the space used plus the estimate is more than the 10 GB Express data-file limit, the update cannot run. The row reads `Database size: SQL Server Express. Database <N> GB of the 10 GB limit; the update needs about <M> GB more and cannot fit. Purge history first (Purge history row below).` If the database is already over the limit with nothing left to estimate, it reads `Database size: SQL Server Express. Database <N> GB is over the 10 GB limit. Purge history first (Purge history row below).` Next is disabled. Purge history (below) is the way forward, or move the database to a SQL Server edition without the 10 GB limit.

2. **Not enough free disk (red, no override), every edition.** If the estimate is more than the free space on the volume that holds the data file, the update cannot run. The row reads `Database size: <Edition>. Database <N> GB, update needs about <M> GB but only <F> GB is free on the data volume. Free disk space or purge history first (Purge history row below).` Next is disabled. Free disk space on that volume, or purge history, then re-run the checks by restarting the wizard.
3. **Free disk could not be read (amber).** If the SQL login cannot read the volume statistics (it needs the VIEW SERVER STATE permission), the row reads `Database size: <Edition>. Database <N> GB, update needs about <M> GB. Free disk space could not be checked (the SQL login cannot read sys.dm_os_volume_stats); make sure the data volume has that much free.` On a database whose 2.5.15 rebuild is already complete it reads `Database size: <Edition>. Database <N> GB, no extra space needed for this update. Free disk space could not be checked.` The update is allowed after a confirmation when you click Next. Check the free space yourself before continuing.
4. **SQL Server Express over 8 GB (amber).** On Express, if the update fits but the database already uses more than 8 GB, the row reads `Database size: SQL Server Express. Database <N> GB of the 10 GB limit; the update needs about <M> GB and fits, <F> GB free. Consider purging history (Purge history row below).` The update is allowed after a confirmation when you click Next.
5. **Otherwise green.** `Database size: <Edition>. Database <N> GB, update needs about <M> GB, <F> GB free.` On a database whose 2.5.15 rebuild is already complete: `Database size: <Edition>. Database <N> GB, no extra space needed for this update, <F> GB free.`

If the measurement itself fails (for example the login lacks rights to the system views), the row is amber and reads `Database size: Could not measure the database (<error>). See the log file.` The update is allowed after a confirmation; the space the update needs is then unknown, so check it yourself.

The two red rules are hard blocks on purpose. A site that fails one of them has, by definition, no room to recover from a half-completed update. The 10 GB and 8 GB thresholds and the factor of 4.5 are fixed and cannot be changed in the wizard.

The recovery model and the transaction log. The print history rebuild changes every Printout record, and a database in the FULL or BULK_LOGGED recovery model keeps all of that in its transaction log until the log is backed up: on a customer database of 24 million print records the log grew from 5 GB to 60 GB. The update therefore switches such a database to the SIMPLE recovery model while it runs and back to its own model when it ends, as the history purge does. While the rebuild is still to run, the Database size row says so: `The database uses the FULL recovery model: the update switches it to SIMPLE while it rebuilds print history and back to FULL afterwards. Take a full backup after the update.` When the update has ended the wizard shows:

> The database was switched to the SIMPLE recovery model while the update rebuilt print history, and is back in FULL. Take a full backup now: transaction log backups cannot continue from a backup taken before the update.

Take that full backup, then let your normal transaction-log backups continue. If you use log shipping, set it up again from the new backup. If the wizard is closed or the server restarts before the update ends, the database is left in SIMPLE: run the update again, and it finishes the work and sets the recovery model back. A database that already uses SIMPLE is not changed and nothing is shown.

SQL Server does not allow the switch for a database that is in an availability group or is mirrored. For such a database the update runs in the database's own recovery model, and the check counts the transaction log as well: about 8 times the size of Printout plus DeletedPrintout, against the free space of the volume that holds the log. The row then reads, when it fits, `The FULL recovery model cannot be switched for the update (availability group or mirroring), so the transaction log will grow by about <L> GB.` , and when it does not fit it is red: `... update needs about <M> GB of data space and about <L> GB of transaction log (the recovery model cannot be switched for the update) but only <F> GB is free on the volume. ...` or, when the log is on a volume of its own, `... update needs about <L> GB of transaction log (the recovery model cannot be switched for the update) but only <F> GB is free on the log volume. ...`

PD licence

Checks that the site's licence is valid, not expired and not revoked, and that maintenance is current. Green reads `PD licence: Licence is valid and Maintenance is up to date.` Any other outcome is red, with the reason in the row (for example `The licence is expired.` or `Maintenance has expired.`), and shows a **New licence** button to load a new licence.

PDAgent .Net versions

Lists PDAgent computers on which .NET Framework 4.8 could not be confirmed. Green reads `PDAgent .Net versions: Confirmed PDAgents have .Net 4.8 installed.` Amber reads `PDAgent .Net versions: Could not confirm PDAgents have .Net 4.8 installed.` with a button that lists the computers. Amber does not block the update; you are asked to confirm when you click Next. PDAgents without .NET 4.8 will not be able to update themselves until it is installed.

Backup database

Reads `Database not backed up` (amber) until a backup is taken through the row's button, after which it reads `Database backup complete` (green). Running the update without a backup is allowed but you are asked to confirm: `Are you sure you want to run the update without backing up the database?` (combined with the .Net confirmation when both apply). If you back up outside the wizard, this row stays amber; that is expected.

A completed backup through this row also satisfies the Purge history option's backup requirement.

Purge history

New in 2.5.15. Always offered, on every SQL Server edition and every database version. Normally shown with a neutral history icon (a clock with a backward arrow) and optional: `Purge history: Optional. Removes print, copy, scan, fax and call history dated before a cutoff you choose (default 2 years) and shrinks the database. Requires a full backup first.` The **Purge history** button opens the purge dialog described in the next section. After a successful purge in this wizard session, the icon becomes a checkmark and the row reads `Purge history: Completed successfully. You can optionally purge more history by choosing another cutoff.`

Two other states:

- **Update in progress (amber, purge unavailable).** A database that is part-way through the 2.5.15 print history rebuild (the update was interrupted) reads `Purge history: Not available until the update finishes: the database is part-way through the 2.5.15 print history backfill. Run the update; the purge is available afterwards.` Next stays enabled so the update can resume. Do not purge such a database by other means.
- **A previous purge did not finish (red, blocks the update).** If a purge left the database in single-user mode or left `Tmp_` tables behind, the row reads `Purge history: A previous purge did not finish: <what was found>. The update cannot run until the database is recovered. Click Copy recovery steps (the steps are also in the log file).` where `<what was found>` is `the database is in SINGLE_USER mode`, `<n> Tmp_ table(s) were left behind (<names>)`, or both joined with `and`. Next is disabled and the button reads **Copy recovery steps**. See Recovering a database after a purge did not finish below.

If the wizard cannot read the database state at all the row is amber, reads `Purge history: Not available: the database state could not be read (<error>).`, and the button is disabled; the update itself is not blocked. The exception is a wizard session in which a purge has already been attempted: the state must then be confirmed before the update can run, so the row is red, reads `Purge history: The database state could not be read after a purge was attempted (<error>). The update cannot run until the state is confirmed: restart the wizard.`, and Next is disabled.

Clicking Next

When every red row is clear, Next runs the update. Amber rows are gathered into one confirmation first. For an amber Database size row, its text is shown followed by `Are you sure you want to run the update?` (or ahead of the backup and .Net question when those apply). Answer No to return to the checks page.

Purging history

Purging history removes old transaction records so that a database fits the update, fits the SQL Server Express limit, or is simply smaller and faster. It is a permanent deletion of the records before the cutoff you choose. The dialog is opened from the Purge history row.

What is removed. Every record dated on or before the cutoff in these tables: Transaction and its children (Copy, Fax, Printout, Scan and PbxCall), PbxRaw, SNMPReading and its children (CounterValues and MarkerSupplies), DeletedPrintout, LastCardScan, RuleViolation and SecurityAreaAudit. The Message table is emptied entirely. Records whose date is not set (SNMPReading and SecurityAreaAudit rows with no date) are kept.

On a database at 2.5.15 or later the print job records themselves (JobSubscription, which holds each job's document name, workstation, queue and paper size) are purged with the history they belong to. A job record is kept when the job is still waiting to be released, whatever its age, when the job was submitted or finished after the cutoff, or when a kept Printout, DeletedPrintout or RuleViolation record belongs to it. Every other job record is removed, along with any release PIN stored for it. On a database below 2.5.15 the job records are not touched (before 2.5.15 that table holds only the jobs currently waiting).

What is kept. Everything after the cutoff, and every configuration table: users, groups, accounts, matters, devices, print queues, rules, prices, licences and settings are untouched. The rebuilt tables are renumbered from 1, so report references to specific record numbers change (job records keep their identifiers).

Which databases can be purged. A database at version 2.2.51 or later, whatever version it is updated from. An older database is refused before anything is changed (the message names the table or column the purge does not fit): update it first, then purge.

How it works. The kept rows are copied into fresh tables, the original tables are dropped, the copies are renamed into place, and the keys and indexes are re-created; then the database is shrunk. This is much faster than deleting row by row, but it needs free disk space of about the size of the kept data while the copies exist, and it is not resumable.

The dialog

While the initial measurements or a new cutoff are being calculated, a message and animated progress bar appear above the table headers. You can select table rows and scroll the list to inspect all tables. The progress log appears when the purge starts; before then, the table uses the available space.

The header shows the date range of the transactions in the database, the number of Printout rows, the current data size and, when relevant, the update's estimated growth against the 10 GB Express limit or against the free space on the data volume. For example: `Transactions from 01/05/2023 to 19/08/2026. Printout rows: 6,163,000. Database 5.4 GB used. The update needs about 6.3 GB of temporary database growth against the 10 GB SQL Server Express limit.`

Cutoff. `Purge history dated on or before:` a date picker, with quick picks for **1 year**, **2 years**, **3 years** and **5 years** back from today. The default is 2 years. Under it: `Everything dated on or before <date> will be purged.`

Projection. Every change of the cutoff recalculates, per table, the rows purged, the rows kept and the current size, and then two lines:

- `About <n> rows will be purged and <m> will be kept. The database will use about <G> GB after the purge.` followed by one of `The update would then need about <M> GB of temporary database growth on the disk holding the database.`, `The update would then need no extra database growth.`, or, when the cutoff is not early enough, `The update would still not fit: <the Database size text as it would read after the purge>`. Choose a later cutoff (less history kept) in that case.
- `The purge itself needs about <K> GB free on the data volume (<F> GB free now).` The kept rows are copied before the originals are dropped. When the free space is short this reads `WARNING: the purge itself needs about <K> GB free on the data volume but only <F> GB is free. Choose a later cutoff (less history kept) or free disk space first.`

The table headings are **Rows to purge** and **Rows to keep**. Update growth means temporary space in the database data files, on the disk holding the database. The dialog states whether there is enough space, or whether free disk space could not be checked. For SQL Server Express, the projected database must also stay within its size limit.

The counts are estimates ("about") from the tables' own date columns; the purge itself uses the transaction date for child records, so the final numbers can differ slightly.

Backup requirement. The **Purge** button is disabled until one of two things is true: the Backup database row on the checks page reports a completed backup (the checkbox then reads `Database backup completed through the checks page` and is locked on), or you tick `I have taken my own full backup of the database`. There is no third option. The purge cannot be undone and a failure part-way through can only be recovered from a backup.

Temporary availability. Clicking Purge shows an information message explaining that the database will be temporarily unavailable, services do not need to be stopped, and PDAgents and PDServer use offline processing where enabled. Other direct database connections are temporarily disconnected. Dismiss the information message to begin; there is no connection list or additional Yes/No warning.

While it runs. The dialog shows the same two progress lines as the execution page (for example `Purging history: running for 46 s. Copying kept rows: [Printout] ...`) and a scrolling log of the purge script's own messages: `Truncating [Message] ...`, `Dropping foreign keys ...`, one `Copying kept rows: [<table>] ...` per table, `Swapping tables into place ...`, then the re-creation of keys and indexes, the shrink, `Rebuilding indexes fragmented by the shrink ...`, `Updating statistics ...` and `=== Purge complete ===`. **Close** is disabled while the purge runs: it cannot be cancelled once started. A database that has the AUTO_CLOSE option on also shows `AUTO_CLOSE turned OFF.` near the start: the purge switches it off for good, as the 2.5.15 update does for every database.

The shrink, the autogrowth settings, the index rebuild and the statistics refresh at the end are housekeeping: by then every kept record, key, index and link is in place. If one of them fails, the log shows a line starting `CLEANUP WARNING:` with the reason, and the purge carries on, hands the database back and reports success.

When it completes the wizard shows:

> The purge completed successfully in .
>
> Database space used: GB before, GB now. Database space recovered: GB. Data-file space returned to disk: GB.
>
> Before running the update, click Backup database to save a full backup of the purged database. Your earlier backup still contains the history that was removed.

If a housekeeping step failed, the message adds `The purged history is complete and the database is back in use, but these housekeeping steps did not finish:` followed by one line per step. No recovery is needed. A failed shrink only means the data file keeps its free space, which SQL Server reuses as the database grows; a failed index rebuild or statistics refresh can be repeated later by a database administrator.

The completion message shows measured space savings before the backup advice. Database space recovered is the decrease in used data space; data-file space returned to disk is the decrease in allocated data-file size, excluding the transaction log. If final measurement is unavailable, the dialog reports success and says the recovered space could not be measured. If used space did not fall, it reports no net reduction rather than negative savings.

The Backup database row reads `Take a new full backup after the purge, before running the update`. Your earlier backup is still valid for restoring the database before the purge. The new full backup captures the smaller, purged database so that an update failure can be recovered without repeating the purge. Keep the earlier backup if you may need the removed history.

For administrators using transaction-log backups: the purge temporarily uses SIMPLE recovery to limit log growth, then restores the original recovery model. Switching back to FULL or BULK_LOGGED requires a new data backup to start a new log-backup chain. A full backup through the wizard provides that baseline; normal transaction-log backups must then continue. Databases already using SIMPLE recovery have no log-backup chain to break, but still benefit from a post-purge restore point. See Microsoft's recovery-model guidance.

If it fails the dialog stays open, the first progress line reads `Purge FAILED. Read the recovery steps below.`, and the log shows the error followed by the script's recovery instructions (the lines starting with `***`). The same text is written to the PDUpdate log file. The database is normally left in single-user mode on purpose. A failure in the setup steps, before anything is copied (the log reads `*** SETUP FAILED before any data was modified`), is different: nothing was purged and no `Tmp_` table exists; the script hands the database back as it was, and if its own recovery lines were cut short the wizard tells you to check the access mode and recovery model instead. When you close the dialog, the checks page runs every check again and reports the database as it actually is. If it was left in single-user mode or with `Tmp_` tables (what a failure after the setup steps normally leaves), the Purge history row turns red, its button changes to **Copy recovery steps**, and Next is disabled until the database is recovered: follow Recovering a database after a purge did not finish below. A setup failure that handed the database back cleanly, or a run that had in fact completed on the server before the connection was lost, shows the row as `Optional` again.

The execution page

After Next, the execution page extracts the component update packages, updates the database, and publishes the packages into the database. The database update shows two lines, each a single fixed line so that the text never spills into the next row of the page:

1. **What is running and for how long**, for example `Migrating to 2.5.15: step 7 of 13, running for 46 s. Adding JobGuid to Printout`. When SQL Server reports progress for the current statement the line also shows `40% complete, about 3 min 12 s remaining`. Durations read `46 s`, `3 min 12 s` or `1 h 12 min`. On the 2.5.15 update, the print history rebuild reports its phase in the same place, for example `Phase 3 of 8: copying print history into JobSubscription parents, 10000 job groups per batch (the longest phase)`.
2. **The SQL statement running**, in a smaller monospace line, trimmed with an ellipsis when long. Hover over it to see the full statement. This line is for support: it lets a screenshot of a long-running step be matched to the update script.

Failed statements, if any, collect in an expander on the same page as they occur. Do not close the wizard while the database update is running. If the update is interrupted (power loss, the wizard killed), run PDUpdate again: the 2.5.15 history rebuild resumes from where it stopped, the checks page shows the Purge history row in its amber “update in progress” state, and Next is enabled.

Recovering a database after a purge did not finish

This section applies when the checks page finds the database in **SINGLE_USER** mode or with `Tmp_` tables left behind. That is what a purge normally leaves when PDUpdate is closed or killed while it is running, or when the script fails after its setup steps: the single-user mode is kept on purpose, so that the Print Director services cannot reconnect to a half-purged database. (A setup failure that unwound cleanly, or a run that completed on the server just before the connection was lost, leaves neither, and the row reads `Optional` again.) When PDUpdate finds this state it reports it on the Purge history row (red), disables Next, and changes the row’s button to **Copy recovery steps**. Clicking it copies the following text to the clipboard and shows it in a message box; it is also written to the PDUpdate log file. The `<database>` placeholder is filled in with the site’s database name.

```
*** The database has been left in SINGLE_USER mode on purpose.

If the failure happened BEFORE the table swap:
*** The original tables are still intact (the failure happened before the swap).
*** To recover: drop all Tmp_ tables, fix the cause, and re-run this script.
*** (Foreign keys may already be dropped; re-running recreates them.)

If the failure happened DURING or AFTER the table swap:
*** The failure happened DURING or AFTER the table swap. Do NOT hand the database
back.
*** To recover: RESTORE the full backup taken before this run, then re-run.

If the script said the purge itself was complete:
*** The purge itself is COMPLETE: the failure happened after every kept row, key,
index and foreign key was in place. Do NOT restore the backup.

*** Once recovered, restore access with:
***   ALTER DATABASE [<database>] SET MULTI_USER;
```

From outside, the wizard cannot tell which of the cases applies, so it shows them all. The third case is told apart by the purge log alone: when it contains `*** The purge itself is COMPLETE`, every data change was made and only the last step, setting the recovery model back or handing the database back, failed. Restoring the backup would throw that work away: run `ALTER DATABASE [<database>] SET MULTI_USER;`, check the recovery model in the database's properties, take a full backup, and run PDUpdate again. For the other two cases: The purge log (in the dialog and in the PDUpdate log file) is a hint, not proof: the last lines can be lost with the connection, so a log that stops before `Swapping tables into place ...` does not by itself prove the swap never started, and a table's name and row count do not prove it is the original (the swap renames each `Tmp_` table into the original's place). Confirm the phase from the `Tmp_` tables in SQL Server Management Studio. The swap starts by dropping `Copy` and renaming `Tmp_Copy` to `Copy`, so the failure was before the swap only when **`Tmp_Copy` and `Copy` both exist**: the rename never happened, and `Copy` is still the original. That is the one state in which the before-swap procedure is safe, and only for a database nobody has touched since the run stopped: if any table has been dropped, renamed, restored or re-created by hand in the meantime, the names no longer prove anything, so restore the backup. In every other state (`Tmp_Copy` missing, `Copy` missing, or no `Tmp_` table at all: a run can fail after the last rename, when no `Tmp_` table is left but the keys and indexes have not been re-created) restore the backup. This also sends a run that stopped while `Tmp_Transaction` was still being filled, before `Tmp_Copy` existed, to the restore: that is deliberate, a restore is always safe and the copying had barely begun. If you cannot confirm the state, restore the backup.

Close PDUpdate before you start: while it is open, its own connection is the database's single user and Management Studio cannot connect.

When nothing can connect. Once PDUpdate closes, the single user slot goes to whichever connection reaches the database first, and the Print Director services and PDAgents keep trying to reconnect. Management Studio then fails with `Database '<database>' is already open and can only have one user at a time`, and a login whose default database is the Print Director database fails with `Cannot open user default database`. To take the slot back:

1. Close Management Studio on every machine connected to this SQL Server instance.
2. Stop the Print Director services on the server (PDServerService, PDServerFx, PDSync) and the PDAgents on any other machine that uses this SQL Server instance.
3. Restart the SQL Server service in Services. This needs Windows administrator rights on the SQL Server machine, not SQL Server rights.
4. Straight away, from a command prompt, connect directly into the Print Director database with the login Print Director uses (a member of the database's db_owner role is enough): `sqlcmd -S <server\instance> -U <login> -P <password> -d <database> -Q "ALTER DATABASE [<database>] SET MULTI_USER;"` (use `-E` in place of `-U` and `-P` for Windows authentication). sqlcmd opens a single connection; Management Studio opens several.

Step 4 hands the database back straight away, which is right only when the purge log says `*** The purge itself is COMPLETE`. In the other two cases, with the services still stopped after the restart, do the steps for your case first: leave out the `-Q "..."` part to keep an interactive sqlcmd session in the database and drop the `Tmp_` tables there, or restore the backup from Management Studio.

Before the swap (the usual case when the wizard was killed while `Copying kept rows` was showing): in SQL Server Management Studio, connected as an administrator, drop every table whose name starts with `Tmp_` in the Print Director database, then run `ALTER DATABASE [<database>] SET MULTI_USER;`. The original tables and their rows are intact; only the Message table, which every purge empties at the start, is gone. The foreign keys between the history tables were dropped before the copy began, and only a purge that runs to completion re-creates them, so do not run the update and do not hand the database back to the services in this state: run PDUpdate again, confirm the Purge history row reads `Optional` again, and re-run the purge to completion. If the purge cannot be completed, restore the full backup taken before it instead.

During or after the swap: restore the full backup taken before the purge, then run PDUpdate again. Do not hand the database back to the services in this state.

In both cases the update is blocked until the row is green or amber again. Support can also run the purge script by hand in Management Studio; contact Blue Swift support if you are unsure which case applies.

Support notes

A PDServer not yet updated logs a foreign key error after the 2.5.15 update

PDUpdate does not stop the Print Director services, and a PDServer on another computer keeps running its old version until it updates itself from the package PDUpdate loaded. Every PDServer before 2.5.15 deletes released print jobs older than seven days from the JobSubscription table on each settings refresh (every 10 minutes) and at startup. From 2.5.15 that table holds the print history, so the 2.5.15 update protects it: each Printout and DeletedPrintout record has a foreign key (`FK_Printout_JobSubscription` , `FK_DeletedPrintout_JobSubscription`) to its job record, and the old delete fails and removes no job record (the release PINs of those old jobs, which nothing uses, are still removed). The old PDServer logs the failure (SQL error 547, `The DELETE statement conflicted with the REFERENCE constraint "FK_Printout_JobSubscription"`) each time it tries, and carries on with the rest of its work; on PDServer versions before 2.3.9 the error also stops the rest of that settings refresh. The errors stop once that PDServer has updated itself. Nothing needs repairing.

From the start of the print history rebuild until it updates itself, a PDServer before 2.5.15 cannot list waiting jobs from the database or record new jobs in it: the update removes columns its queries name, before it writes any history the old PDServer could mistake for waiting jobs. Jobs the old PDServer already holds in memory are still listed. A job the old PDServer stores itself in that time (an email-to-print or cloud print job held on the server) is kept only in its memory, so it is lost when that PDServer restarts or updates before the job is released. Update every PDServer as soon as the database update has finished. If the update stops part-way, those PDServers stay in that state until the update is run again.

The history purge (Purge history, or `ClearTransactions-TmpRebuild.sql` run by hand) refuses a database in which a Printout or DeletedPrintout record has lost its job record, with `Printout / DeletedPrintout hold rows whose JobSubscription parent is missing. The database was NOT purged and remains usable. ...` That can only happen on a database updated to 2.5.15 before these keys existed and then swept by an old PDServer. Contact Blue Swift support: `Tools\SupportScripts\RepairS5BackfillOrphanParents.sql` restores the job records of Printout rows from the backup taken before the update (it reports DeletedPrintout rows without repairing them).

ClearTransactions-TmpRebuild.sql and RuleViolation.JobGuid (fixed 2026-09-05)

The 2.5.15 database update adds a `JobGuid` column to the RuleViolation table, linking a rule violation to the print job that caused it. Until 2026-09-05 the support purge script `ClearTransactions-TmpRebuild.sql` (the same script the wizard's Purge history option runs) built its temporary `Tmp_RuleViolation` table without that column, so a 2.5.15 database purged **by hand with the old copy of the script** lost `RuleViolation.JobGuid` on every kept row. The column is nullable and only feeds the 2.5.15 rule-violation-to-job link; nothing else reads it, and the rows themselves are intact. No repair is possible because the values are gone; the link is simply absent for violations recorded before such a purge.

The wizard's built-in purge was never affected: it checks the script's column list against the live tables before touching anything and refuses to run on a mismatch. Only a manual run of the pre-2026-09-05 script against a 2.5.15 database is affected. Make sure any copy of the script kept on a customer site is the current one from the Print Director 2.5.15 support tools.

Print Director version at time of publish: 2.5.15.3. Publish date: 29/10/2026.