How does Exalate prevent sync loops?

Last Modified on 02/17/2026 9:32 am EST

How does Exalate prevent sync loops?

Exalate prevents **sync loops** by employing built-in **change detection** that differentiates between updates made by users in the local system and those originating from synchronization. This approach ensures that changes received from synchronization don't trigger additional sync cycles, keeping systems in sync without creating loops.

Change detection for sync loops

When an **incoming script** applies changes to a work item, Exalate marks those modifications as **sync-generated**. This allows the **outgoing script** on the same side to detect the sync-generated changes and prevent another synchronization cycle from being triggered.

This mechanism ensures that only **genuine local changes** (made by users) initiate an outgoing synchronization to the connected platform. Changes that came from an incoming sync do not loop back and cause redundant synchronization.

The role of the firstSync flag

The **firstSync flag** is crucial for preventing sync loops during **initial synchronization**. This flag helps distinguish between a **first-time sync** and subsequent updates. When a work item is synced for the first time, you can apply different logic, treating the initial setup differently from ongoing updates.

This is particularly helpful for preventing loop-related issues that often arise during **initialization** or setup behavior.

Timestamps and version tracking to avoid conflicts

Exalate uses **timestamps** and **version tracking** to detect when both sides have been updated since the last sync. If changes happen simultaneously on both sides, Exalate processes them **sequentially**, ensuring data consistency and avoiding competing update cycles.

Your scripts can include **conditional logic** to decide how to handle simultaneous changes, such as:

- **Preferring the most recent update**
- **Requiring manual resolution** for specific fields

Sync queue mechanism for orderly processing

The **sync queue** mechanism is another key element of Exalate’s loop prevention. Exalate tracks **pending operations** to ensure changes in flight are processed in order. This prevents reprocessing of updates that are already queued or recently completed, ensuring **orderly synchronization** even when rapid changes happen across connected systems.

Conditional logic for conflict resolution

You can configure **conditional logic** in your scripts to handle potential conflicts, especially when changes occur simultaneously on both sides. This gives you **fine-grained control** over how conflicts are resolved, ensuring the right data is prioritized and reducing the chance of synchronization errors or data loss.

This approach to **sync loop prevention** ensures smooth, reliable, and efficient synchronization without redundant updates, even in complex scenarios with high-frequency changes.