SM36 AND SM37 AT ENTERPRISE SCALE: WHEN SAP BACKGROUND PROCESSING BECOMES CROSS-SYSTEM ORCHESTRATION

What SM36 and SM37 do

SAP ships with a complete, native toolset for background processing. SM36 is where a job is defined: its steps, program variants, start conditions, and recurrence. SM37 is the job overview, where administrators monitor, analyze, and troubleshoot runs. Two companion transactions manage the events that trigger them: SM62 and SM64 handle event definition and administration.

These transactions are well designed for the work they were built for, and that work is substantial. A team can schedule an ABAP program to run nightly, chain steps in sequence, and review the log the next morning — reliably, with no additional software and no additional interface to learn. For single-system batch processing, native SAP tooling is the right answer and remains so.

What changes is the shape of the work. As a business grows, a background job stops being a standalone task and becomes one step in a business process that spans SAP and the systems around it: carriers, banks, data platforms, partner networks. The transaction still runs the job correctly. What the business now needs to see is whether the *process* completed. That is a different question, and it is where an orchestration layer extends what SAP already provides rather than replacing it.

Four places make that shift concrete.

1. Dependency logic: from linear chains to process flows

SM36 expresses mainly sequential dependencies: run step B after step A. That models a linear chain accurately. Enterprise business processes rarely run in a straight line — they fan out into parallel branches that rejoin, and they carry logical dependencies where a successor runs only if a condition is met. Native scheduling has no integrated way to express this and, just as importantly, no way to *display* it. Without a flowchart of the run, the shape of the process lives in documentation and in the knowledge of the people who built it.

An orchestration layer models the whole process as a visual flow, with sequential, parallel, and logical dependencies in one view. The dependency graph becomes something a team can read, hand off, and audit rather than reconstruct.

2. Spawned and child jobs: seeing thewhole processtree

Many SAP programs spawn child jobs at runtime. SM36 and SM37 have no reliable detection of these spawned processes, so a parent job can report success while its children are still running. The available workaround is to pad the schedule with buffer time and let the children finish before the next step begins. That buffer is idle runtime on a good night and a broken dependency on a bad one.

Child-tree detection removes the estimate. When the full parent-child tree is tracked, the next step in the process starts the moment every predecessor child completes — not on a timer, and not on an assumption.

3. Monitoring across the estate: from after-the-fact toahead-of-time

SM37 reports on the system it runs in, which is exactly its design. At enterprise scale, that means an administrator opens SM37 in each system and reads logs after the run. Failures and delays surface when someone looks; a process can hang in a yellow state without a timely signal; and there is no forward prediction that a run is trending toward a missed deadline. Processes that cross system boundaries—an SAP run feeding a carrier, a bank, or a business intelligence platform—sit outside SM37's field of view, because SM37 sees SAP.

This is where a service level agreement (SLA) becomes the unit of management. Central orchestration watches the whole estate from one place, alerts on a stalled process before it cascades—including when background work is starved because dialog processes are consumed with no free background process—and predicts an SLA breach early enough to act on it.

4. Governance at scale: bringing every process under one view

Anything scheduled directly in SM36 is, by default, outside central control. Individual teams create jobs on individual systems, and no single view holds the complete picture of what runs where. Two mechanisms close that distance. **Interception**, using SAP's certified External Interface for Background Processing (XBP) 3.0, holds user-scheduled jobs so conditions and dependencies can be applied before they run. **Extraction** mirrors those unmanaged jobs into central governance, so the real estate is visible and managed rather than assumed.

The recurring housekeeping layer

Much of what keeps an SAP landscape healthy is recurring maintenance that runs through these same transactions and gets watched by hand. The inventory below is representative rather than exhaustive, and it is the work that scales least gracefully as landscapes grow.

Transaction	Housekeeping it runs	Why it needs watching
SM37 / RSUVM008	Cleanup of stale lock entries	Buildup degrades performance if the run fails unnoticed
SWUI	All SW* workflow and work-item jobs	Individually monitored, buried in system logs
SOST	SAPconnect email, message, and alert delivery	Delivery failures need manual escalation to the right team
RZ11	Intermediate document (IDoc) push, purchase-order and invoice flag updates, maintenance jobs	Silent failures ripple into downstream postings
PFCG / SU01	Recurring security and governance, risk, and compliance (GRC) runs	Authorization and compliance runs must stay on schedule
RZ20	Computing Center Management System (CCMS) alert aggregation	Alerts sit in per-system silos with no central view
SARA	Data archiving: write, delete, and store runs	Estate bloat if archiving lapses; each object tracked by hand

Every row is work someone monitors, escalates, and restarts by hand. Consolidating it under one control point is the difference between a team that babysits background processing and one that manages the business process by exception.

Frequently asked questions

What are the main limitations of SM36 and SM37 at enterprise scale?

SM36 and SM37 are well suited to defining and monitoring background jobs within a single SAP system, and they do that job reliably. Organizations reach past them when those jobs become steps in larger business processes that span systems. Four things drive the shift: SM36 expresses mainly sequential dependencies, with no visual flow and no support for parallel or logical branches; it cannot reliably detect spawned child jobs, so teams pad schedules with buffer time; SM37 monitoring is system-by-system and after the fact, with no forward SLA prediction and no visibility into non-SAP steps; and anything scheduled directly in SM36 sits outside central governance. Closing that distance calls for an orchestration layer that models the full process, detects the parent-child job tree, monitors the estate against SLAs from one point of control, and brings unmanaged jobs under governance through interception and extraction.

Can SAP background jobs be scheduled across SAP and non-SAP systems?

Not natively. SM36 schedules and SM37 monitors work inside the SAP system where they run, so a process step that lands in a carrier portal, a bank file transfer, a data platform, or a partner system is invisible to them; the SAP job reports success, and what happens next is tracked somewhere else, usually by a person. An orchestration layer that integrates through SAP's certified XBP interface

places SAP and non-SAP steps in a single dependency graph, so the end-to-end business process is defined, monitored, and recovered as one unit instead of a set of handoffs.

How do you monitor SAP background processing across multiple systems from one place?

SM37 gives a per-system view, so multi-system landscapes mean opening SM37 in each system and reading logs after the run, with CCMS (RZ20) alerts sitting in their own per-system silos. A central orchestration layer aggregates the estate into one point of control: active status across every connected system, alerting on stalled or starved processes as they happen, SLA prediction that flags a run trending toward a missed deadline, and captured job logs and spool output routed to the team that owns the process. The practical shift is from reading logs after a failure to being told before a deadline is missed.

Next steps

If your background processing has outgrown a single system, the next step is seeing how SAP and non-SAP jobs sit in one dependency graph. Explore Control-M for SAP or access the Control-M demo library.