Log-On SOFTWARE

Db2 FOR z/OS | UTILITY AUTOMATION

Total Utility Control for Db2 z/OS

RETURN ON INVESTMENT

TUC Version 2 Release 5

A Technical White Paper

Log-On Software

Total Utility Control for Db2 z/OS – Return on Investment

TUC Version 2 Release 5

This document is published by Log-On Software for informational purposes. It describes the return-on-investment characteristics of Total Utility Control (TUC) for Db2 for z/OS environments.

The information contained in this document is provided for general guidance and reflects the capabilities of TUC as of the version referenced on the cover page. Product capabilities are subject to change.

Db2 and z/OS are trademarks of International Business Machines Corporation (IBM). Total Utility Control and TUC are products marketed by Log-On Software. All other trademarks are the property of their respective owners.

www.log-on.com

CONTENTS

CONTENTS 3
EXECUTIVE SUMMARY 5
ENHANCING DATABASE AND UTILITY EXECUTION PERFORMANCE 6
Measuring the signal that actually matters 6
Proving the performance payoff, not just assuming it 6
Putting improved access paths to work immediately 7
Keeping the optimizer's information sharp between reorgs 7
REDUCING RESOURCE AND INFRASTRUCTURE COST 8
Smarter backups with incremental copies 8
Collecting statistics only when needed 9
Running REORG only when it pays 9
Reclaiming space with compression 9
Trimming old data to cut storage and improve access performance 10
Refreshing test environments for a fraction of the cost 10
PROTECTING AVAILABILITY AND RECOVERABILITY 11
Guaranteed recovery points 11
Flexible, fast recovery 11
Preventing outages before they happen 12
CENTRALIZING CONTROL OF MAINTENANCE 13
A single, centralized framework manages every utility execution 13
Bringing existing jobs under control without changing them 13
Protecting performance when access paths change 14
Running maintenance safely alongside everything else 14
FREEING UP THE DBA TEAM 15
Set it up once, then let it keep itself current 15
Maintenance that decides for itself 15
Doing big jobs faster with parallel processing 16
Getting the payoff from IBM Db2 Data Analytics Accelerator without the hassle 16
PROTECTING YOUR INVESTMENT OVER TIME 17
Vendor independence 17
Keeping pace with new Db2 versions 17
Fast implementation and low total cost of ownership 17
Deploying across Db2 subsystems 17

MEASURING THE RETURN 18
ABOUT LOG-ON SOFTWARE 19

EXECUTIVE SUMMARY

Total Utility Control (TUC) automates the everyday maintenance of Db2 for z/OS databases such as reorganizations, copying/recovering, and statistics collection - so that this work is done only when it is needed, rather than on a blanket schedule, and in the most efficient possible manner. By basing every decision on metrics such as how much an object has changed, along with the impact of that change, TUC cuts the cost of routine maintenance while improving the recoverability and availability of each database. The payoff is better database performance, lower CPU consumption, fewer outages, and less manual effort for the DBA team. The sections that follow describe where TUC delivers a measurable return, grouped by the type of value provided: enhanced performance, reduced cost, improved availability, centralized maintenance control, DBA productivity and investment protection.

TUC Delivers Measurable Returns

Value Area	What It Means
Enhancing Database Performance	Reorg, runstats, and rebind triggered by real access-performance signals - and verified afterward to prove the performance gain.
Reducing Resource and Infrastructure Cost	Lower CPU, storage, and elapsed time by performing maintenance only when it is genuinely needed.
Protecting Availability and Recoverability	Verified, guaranteed recovery points and fast, flexible recovery when something goes wrong.
Centralizing Control of Maintenance	Authorized, audited execution queue integrated with your enterprise scheduler and driven by your existing job flow.
Freeing Up the DBA Team	Policies that decide for themselves, keep pace with a changing environment, and scale automatically all under the centralized supervision of the DBA team.
Protecting Your Investment Over Time	Utility vendor independence, fast implementation, and low total cost of ownership over time.

ENHANCING DATABASE PERFORMANCE

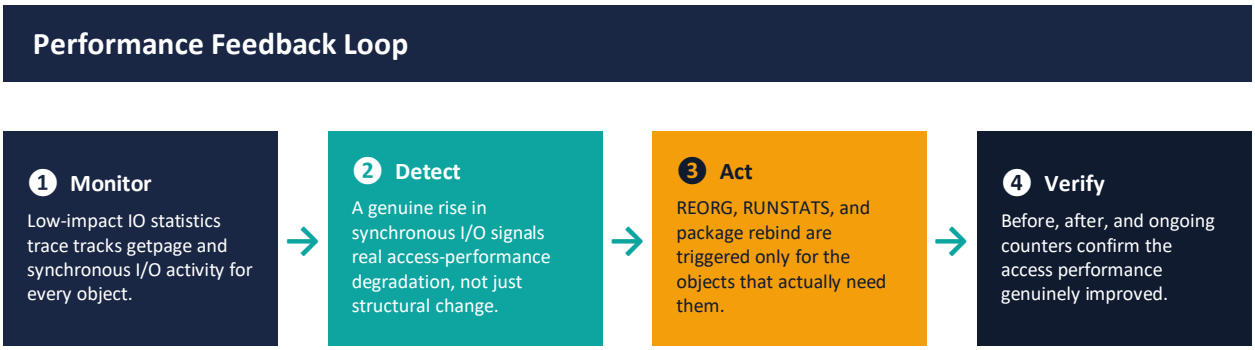


Figure 1. TUC closes the loop between detecting a genuine performance problem and confirming it was fixed.

Database performance ultimately comes down to how efficiently Db2 can get to the data it needs, and that depends on access paths staying well-tuned as tables and indexes evolve. Structural measures like disorganization or data growth are useful early warnings, but they do not always mean application performance is actually suffering. TUC goes a step further: it measures real access performance directly, acts only where that performance has genuinely degraded, and then measures again to confirm the fix worked – turning performance tuning from a guess into a closed, provable loop.

Measuring the signal that actually matters

Structural wear and tear on a table is easy to see, but it does not always translate into a real performance problem for the applications that use it. TUC measures the thing that does: how efficiently Db2 is actually reading each object. It collects IO statistics through Db2's own statistics trace, extracting the getpage and synchronous I/O counters that reveal whether access performance is genuinely degrading. Because this trace produces very few SMF records per interval, the visibility comes at negligible cost, and the trace can be started automatically whenever Db2 comes up. Objects that look disorganized but show no real rise in synchronous I/O are left alone; only those genuinely struggling to deliver good access performance are put forward for action.

TECHNICAL NOTE

TUC turns raw getpage and synchronous I/O counters into a single before/after ratio for each object – a rising ratio flags real degradation, a falling one confirms the fix worked.

Proving the performance payoff, not just assuming it

Running a reorganization or refreshing statistics is only half the job; knowing whether it actually helped is the other half. TUC keeps a continuous record of access performance for every object – what it looked like before the last maintenance action, immediately after, and currently – so the effect of every REORG is visible rather than assumed. Over time this turns performance tuning from an act of faith into a measured, evidence-based practice.

Putting improved access paths to work immediately

Left as a manual follow-up step, rebinds are easy to miss – and an application can keep running on a stale access path even after the underlying object has been reorganized. TUC closes that gap automatically: once a REORG completes, it generates rebind statements for every valid, dependent, and trigger package, so that, upon DBA review and approval, applications can begin benefiting from the improved access paths right away, with no separate step for the DBA team to remember.

Keeping the optimizer's information sharp between reorgs

Access performance depends as much on Db2 having current, trustworthy statistics as it does on the physical state of the data itself. Beyond triggering RUNSTATS only when data has genuinely drifted, TUC keeps a continuous watch for the signals that current statistics may no longer be serving the optimizer well: it can pick up Db2's own autonomic monitoring alerts, and it can watch for updates to an object's statistics-collection profile so it stays consistent with the optimizer's own feedback over time. The result is a set of statistics that stay quietly aligned with how Db2 is actually choosing to access the data, rather than drifting out of step between scheduled collections.

KEY TAKEAWAY

Every performance action TUC takes is measured before it runs and verified after it completes – so the return on every REORG, RUNSTATS, and rebind is a number, not a guess.

REDUCING RESOURCE AND INFRASTRUCTURE COST

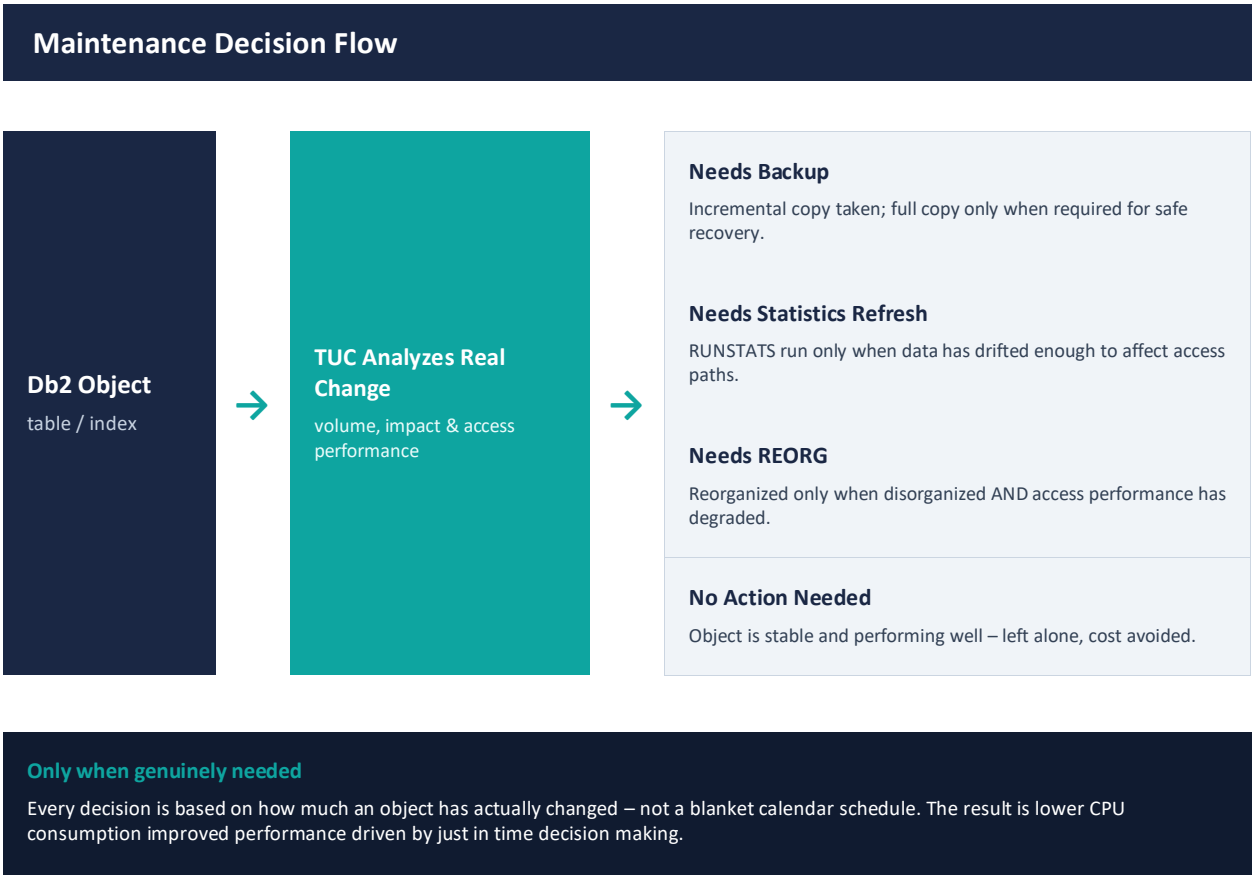


Figure 1. TUC bases every maintenance decision on how much an object has actually changed.

Smarter backups with incremental copies

Database backups must be recoverable to specific points in time that have meaning to the business. The simplest approach - a full backup of everything, every day - is also the most expensive in CPU time, elapsed time, and storage. There is a well-known, far cheaper alternative: incremental backups. Incremental backups capture only the data that has changed since the last backup and therefore run much faster and use far less storage. The catch is that doing incremental backups safely is genuinely complex. You must constantly judge which objects have changed enough to justify a full copy while ensuring that incremental copies required for recovery are all readily available. Because of this complexity, many shops have never adopted incremental backups and keep paying, year after year, the cost of daily full backups. This is a cost-savings opportunity TUC unlocks. It manages the entire incremental strategy for you: it tracks how much each object has changed, takes incremental copies when it is the smart choice, automatically falls back to a full copy whenever one is required for safe recovery, and forces periodic long-term backups on a calendar basis. You capture the large savings of an incremental backup strategy with none of the risk of the hands-on management that normally prevents incremental backup adoption.

Collecting statistics only when needed

Db2 decides how to access your data based on statistics that describe the size and shape of each table and index. Keeping these statistics current matters for performance. The typical approach is to recollect them on a regular basis. But re-collecting statistics for data that has barely changed wastes elapsed time and CPU time. TUC takes a smarter approach: it looks at how much each object has actually changed and refreshes statistics only when the degree of object change indicates they are genuinely out of date. It also watches for signs that statistics have drifted and refreshes them automatically when needed. The result? Statistics remain accurate for all Db2 objects while the cost of collecting them falls sharply.

Running REORG only when it pays

Over time the rows in a table become scattered, and the table needs to be reorganized to restore good performance and reclaim space. The hard part is deciding which objects genuinely need it. Reorganizing all objects on a fixed schedule wastes large amounts of CPU processing objects that are perfectly fine; reorganizing objects too infrequently lets database performance quietly degrade. TUC removes the guesswork. It identifies the objects that have become disorganized and reorganizes only those, handling the whole process automatically.

> **KEY BENEFIT**
>
> *TUC keeps a running total of the processing time it has saved by not running unnecessary reorganizations – turning a maintenance judgment call into a number you can point to.*

It goes a step further. Some objects look disorganized by the usual structural measures yet are still performing perfectly well, so reorganizing them again spends processing time to no benefit. TUC watches the real access performance of each object and recommends reorganization only when the object is both disorganized and showing performance degradation. An object that is fragmented but performing well is left alone. Every reorganization avoided is a real, repeatable savings. TUC keeps a running total of the processing time it has saved by not running unnecessary reorganizations. This turns a maintenance judgment call into a number you can point to.

TUC's ability to automatically identify objects in need of reorganization just as that need arises, and for that select set of objects automatically execute the reorganization, turns a periodic large-scale effort into an incremental effort that can be executed daily. The backlog of objects in need shrinks to those objects newly in need that day. Just-in-time reorganization further improves performance across all databases.

Reclaiming space with compression

Compressing a table lets it hold the same data in less space, which means less disk, smaller and faster backups, and less logging. The question is whether compressing a given table will save sufficient storage, as well as sufficient logging, backup, and recovery time and CPU, to justify the compression CPU cost. TUC answers that automatically: before compressing, it estimates if the cost of compression is worth the gain. It then selects the

most effective compression approach, compresses the tables where there will be a net benefit from compression and leaves the others alone.

Trimming old data to cut storage and improve access performance

Smaller tables are cheaper to store, faster to access, faster to reorg, and more economical to maintain, yet data sitting in production tables is often old and rarely touched. TUC makes it easy to purge or archive that old data using conditions you define for each table, and it keeps a record of whatever is removed so it can be reloaded later if needed, either into the original tables or elsewhere. Keeping only the data you use in your production tables improves and lowers CPU and storage costs.

Refreshing test environments for a fraction of the cost

Populating test and development environments with a fresh copy of production data is a routine but costly chore. The usual approach reads and rewrites every single row, which burns a great deal of processing and elapsed time. TUC offers a far cheaper path: it copies the data directly using the DSN1COPY utility, which is typically ten times faster and less expensive. You simply point TUC at the production data and at the target Db2 subsystem, and it works out on its own which production data belongs to which target table or index, even when their names differ, and confirms the two are compatible before copying anything. Anything that cannot be copied directly is reported so it can be adjusted or handled separately, so nothing is ever copied incorrectly.

When refreshing tables, the largest saving comes from not repeating work that has already been done. On most recurring refreshes, many partitions may be unchanged since the last copy. TUC detects which partitions have changed, copying just those partitions and not the unchanged ones. This keeps the cost of every refresh low and makes it practical to refresh far more often, giving developers and testers current data on demand. An entire test environment can be refreshed in a single run, largest objects first, with the work spread across parallel jobs, and a summary shows how much extra space the refresh needs before it starts. The same mechanism can also restore a dropped object or copy an older backup into side tables so the data can be examined as it stood at an earlier point in time.

In summary, TUC masks all the complexity of setting up DSN1COPY copy jobs making its routine use not only cost effective, but practical.

PROTECTING AVAILABILITY AND RECOVERABILITY

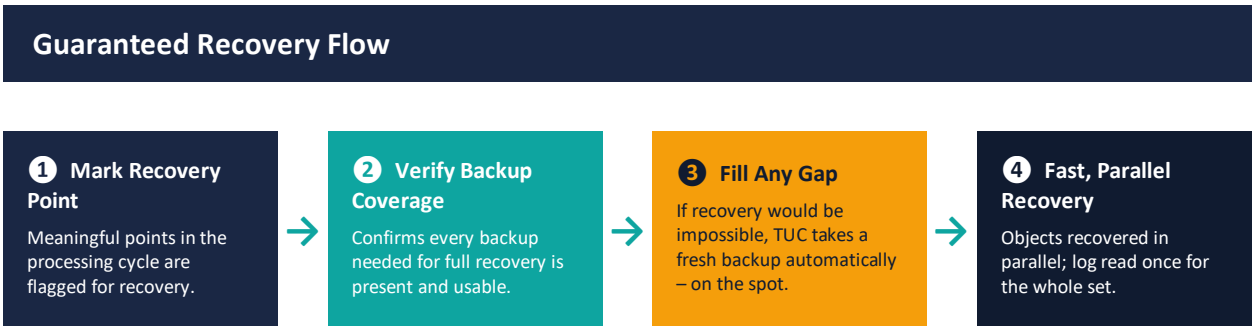


Figure 2. Guaranteed recovery points are verified, not assumed.

Downtime affects operations, revenue, and reputation, and there are far too many possible failure scenarios to prepare for each one by hand. The database team needs tools that deliver the right recovery quickly and, just as importantly, guarantee that what was backed up can be restored.

Guaranteed recovery points

When something goes wrong - a bad batch run, a corrupting application error - you want to roll the data back to a known-good moment. The worst outcome a database shop can face is to discover, in the middle of that crisis, that the backup it is counting on is unusable or that a required backup is missing. If the data cannot be recovered - an otherwise manageable incident may turn into an outage stretching into hours or days. TUC mitigates that risk. It lets you mark meaningful recovery points in your processing cycle and guarantees they are usable for full recovery. It automatically checks that nothing has happened since the last backup that would make recovery impossible, and if it finds such a gap it takes a fresh backup on the spot. It also confirms that every backup needed to recover up to the present moment is available. Your recovery points are verified rather than assumed, providing assurance you can recover before you need to.

> **TECHNICAL NOTE**
> *TUC automatically checks that nothing has happened since the last backup that would make recovery impossible – and if it finds such a gap, it takes a fresh backup on the spot.*

Flexible, fast recovery

TUC generates recovery jobs for any group of objects through a simple dialog: a single table, an entire database, a whole disk volume, back to the last backup, or to a chosen point in time. When you recover to an earlier point, it restores the related indexes from their backups instead of rebuilding them from scratch, which is far faster, and it can pull in dependent objects automatically, so the data stays consistent. It speeds recovery further by handling many objects in parallel and reading the log just once for the whole set. For disaster recovery, recovery jobs can be prepared in advance and run in the order of priority, so your most critical data is made available first.

Preventing outages before they happen

Many outages are avoidable if you can see them coming. TUC continuously watches how fast each object is growing and warns you before one is about to run out of space, so you can act in advance instead of reacting to a failure in the middle of the night. It also watches for objects slipping into restricted or unusable states and automatically prepares the corrective action needed to put them right. Heading off these problems not only avoids downtime but keeps the whole system running with more headroom.

CENTRALIZING CONTROL OF MAINTENANCE

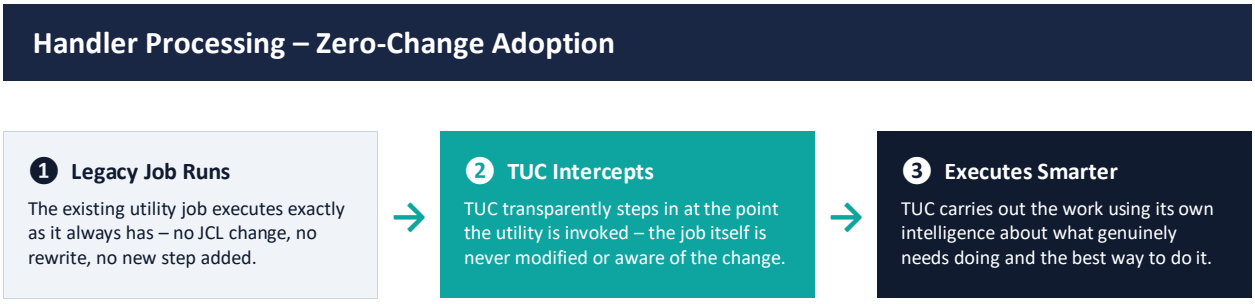


Figure 3. Handler processing brings legacy utility jobs under TUC control with zero change.

A single, centralized framework manages every utility execution

In many shops, utilities are run ad hoc from scattered jobs, with no central oversight of what runs when or who is allowed to run it. TUC channels every utility through one controlled request queue. Maintenance can be requested by the DBA team, by an application at the right point in its own processing, or by other authorized users - but it always runs under the framework and the DBA team's control, with full visibility into what was requested, by whom, and how the request was resolved. This gives you one place to enforce who may run powerful or disruptive operations while maintaining a complete audit trail of all maintenance activities. The benefits TUC's centralized control provides extend beyond efficiency - TUC heads off the expensive mistakes that scattered, uncontrolled utilities invite: a destructive utility aimed at the wrong objects, or a disruptive one started in the middle of peak processing. These mistakes can lead to an outage, an emergency recovery effort, and hours of firefighting. Funneling every utility through a single authorized, audited queue removes that whole class of costly accidents, along with the overhead of chasing down who ran what. The DBA team gains non-disruptive control without becoming a bottleneck.

KEY TAKEAWAY

TUC channels every utility through one centrally managed request queue, with full visibility into what was requested, by whom, and how the request was resolved.

Bringing existing jobs under control without changing them

Most shops have many utility jobs running throughout their batch schedule, often independently defined by application teams over years. Bringing these under the control of an automation framework normally means rewriting job after job - a large, risky project that most organizations refuse to take on. TUC removes this barrier. It can transparently take control of an existing utility job without any change to the job itself: when the job runs, TUC steps in and carries out the work defined by the job, but in its own smarter way, applying TUC intelligence about what genuinely needs doing and how best to do it. You can adopt TUC's benefits across your existing workload gradually, one job at a time, with no rewrites and no disruption - an easy non-disruptive on-ramp rather than a rip-and-replace.

Protecting performance when access paths change

After an object is reorganized and its statistics refreshed, Db2 may recalculate the access path it uses to reach the data. Usually that produces a better access path, but occasionally it produces a worse one, and an unexpectedly poor access path can quietly cripple a critical query. TUC manages this risk. When it refreshes access paths after maintenance, it can compare each new access path with the previous one and either block any change that would alter the access path - keeping things stable for routine maintenance - or present access path changes for review, allowing the administrator to selectively promote beneficial changes. It can also be conservative about the conditions under which it recalculates, doing so only when every related object is in good shape, so access paths are recomputed only from a position of strength. You get the optimization opportunities offered by regular maintenance without the risk of a surprise performance regression.

ADMINISTRATOR TIP

You get the performance optimization opportunities made possible by regular maintenance without the risk of surprise performance regressions – access path changes can be blocked or held for review.

Running maintenance safely alongside everything else

From with your legacy utility jobs, TUC transparently produces ready-to-run maintenance requests that drop straight into the job scheduler you already use. There is nothing new to learn and no change to how you schedule work. A TUC maintenance request inherits the very same scheduling dependencies as the job that asked for it, so it continues to run in the right place in the batch flow - after its predecessors and before its successors - instead of cutting in line and colliding with application work. When maintenance is part of an application's overnight batch flow the application can wait for the maintenance request to finish and then carry on automatically. Before releasing a maintenance request to the scheduler, TUC also checks whether the objects it needs are currently in use by something else. If they are being held, it does not force its way in and fail. Instead, it postpones the work and automatically picks a better moment. The payoff is fewer utility execution failures or contentions. Instead, maintenance quietly works around the business rather than disrupting it.

To further the stability of TUC-driven maintenance, TUC governs utility execution throughput. It can cap how many jobs run at once, so a burst of maintenance never overwhelms the system, and it gives urgent, explicitly requested work precedence over routine background work, so applications are never left waiting behind a long queue. For space-hungry operations it confirms that enough free disk exists before each job starts, sets that space aside, and holds back any job that will not fit - letting it run the moment space frees up, so jobs complete in an efficient order instead of failing partway through and having to be rerun. And when a job that was waiting on a dependency is running late and the risk of collision has passed, TUC releases it rather than leaving it stalled. Together these controls turn what can become a chaotic set of maintenance requests into a smooth, predictable, high-throughput flow.

FREEING UP THE DBA TEAM

Balanced Parallel Job Scheduling

Pending Work (sorted by size)

Small	Small	Medium	Large	Small	Medium	Large

TUC deliberately mixes job sizes across streams so no stream is starved of disk space:

Stream 1	**Large**	**Small**	**Medium**	
Stream 2	**Small**	**Large**	**Medium**	
Stream 3	**Medium**	**Large**	**Small**	

Figure 4. TUC mixes job sizes across parallel streams to keep throughput high and disk pools turning over.

Beyond saving machine resources, TUC saves the most expensive resource of all: the time and attention of your database experts. It does this by automating the low-level but necessary repetitive judgment calls and manual tasks that distract from more important work.

Set it up once, then let it keep itself current

Databases are never static - new tables, indexes, and whole databases appear constantly as applications evolve. With hand-built maintenance, every additional object is one more thing someone has to remember to wire in, and anything overlooked silently goes unmaintained and unprotected. TUC removes that ongoing burden entirely. Once your maintenance approach is set up, TUC keeps up with the changing environment automatically: it creates maintenance coverage for any newly created database, refreshes the underlying jobs so they automatically include new tables and indexes, and tidies up after objects that are dropped. Because new objects are covered the moment they appear, they are immediately considered for backups, statistics collection, and reorganization without anyone lifting a finger. It also prepares each maintenance job once and reuses it on every routine run, regenerating itself only when a change has an effect on a maintenance job, so staying current never means redoing work from scratch. The practical effect is that the DBA team configures things once and then monitors rather than manages. The environment can grow to tens of thousands of databases with TUC-drive maintenance simply keeping pace, with nothing slipping through the cracks.

KEY BENEFIT

The environment can grow to tens of thousands of databases. TUC-driven maintenance simply keeps pace, with nothing slipping through the cracks.

Maintenance that decides for itself

You have your maintenance policies - how often to back up, how long to keep copies, when to refresh statistics. TUC turns these policies into automatic decisions. It continuously checks each object against your rules and picks out exactly which objects need attention, so maintenance reaches the objects that need it, when they need it, and skips the ones that do not. If you choose, you can review and fine-tune what TUC selects before anything runs, setting aside objects that need special handling, and let it schedule the balance of the work automatically. As your confidence in TUC grows, you can minimize the review process and let TUC does what it does best -- automate and optimize all utility executions.

Doing big jobs faster with parallel processing

When a large amount of maintenance has to run, TUC can spread the work across many jobs running at once, automatically dividing the objects into balanced groups so the jobs progress and finish evenly instead of leaving one job grinding away for hours. It coordinates the pieces that must happen together and signals to your application flow when the whole set is safely processed. Maintenance windows shrink, and you get more done in less time.

Running large reorganizations in parallel requires sophisticated space management. While it runs, each large reorganization needs a sizeable block of temporary workspace. If too many large jobs run at the same time, they can exhaust the shared disk pool and fail - even though plenty of space would have been free as soon as some of them completed. TUC avoids this by deliberately mixing job sizes. It sorts the pending work into small, medium, and large jobs and schedules a blend of them together rather than starting all the big ones first. The quick small and medium jobs finish fast and hand their space back for the next large job, so the disk pool keeps turning over and stays evenly used. The result is sustained high throughput, without the out-of-space failures that would otherwise occur when many large reorganizations run at once.

Getting the payoff from IBM Db2 Data Analytics Accelerator without the hassle

IDAA can dramatically cut the cost of heavy analytical and reporting queries. Instead of running them on the mainframe, where they consume large amounts of processing power and disk I/O - the most expensive resources - the accelerator runs them on IFLs faster and at a lower cost. The savings are real and substantial, but they only materialize if applications can trust the accelerated copy to be current. If the data might be stale, queries fall back to the mainframe and the savings evaporate. Keeping that copy reliably in step after every utility-generated change is an ongoing chore, and the risk of getting it wrong is enough to stop many shops from leaning on the accelerator for critical work. TUC removes that burden. It keeps the accelerated copy in step automatically whenever a maintenance job such as a REORG DISCARD changes the data, and it is smart about it, refreshing only changed objects (e.g., a specific table partition) and skipping those with no changes (e.g., unchanged partitions). You capture the accelerator's full processing and I/O savings with the peace of mind that the data is always current - and without taking on a new maintenance headache.

PROTECTING YOUR INVESTMENT OVER TIME

Vendor independence

Db2 utilities can be licensed from several different vendors, and the differences in price and capability between them are significant. Mixing vendor utilities to take advantage of the best in class is easy with TUC. . TUC sits above the utilities and works equally well with all of them, so you can mix vendors or move between them smoothly to pursue the best deal or best set of capabilities, without destabilizing your TUC-driven maintenance automation.

> **BEST PRACTICE**
> TUC sits above the utilities and works equally well with all of them, so you can mix vendors or move between them smoothly to pursue the best deal or best set of capabilities

Keeping pace with new Db2 versions

Each new release of Db2 brings features that can require updates to any home-grown maintenance procedures. TUC keeps pace with the latest Db2 releases automatically, adapting to new capabilities as they arrive, so you can take advantage of them right away without re-engineering your own automation.

Fast implementation and low total cost of ownership

Rolling out an automation framework can be a project in itself. TUC installs easily and arrives with sensible 'IBM best practice' defaults that often fit as they are, needing only minor adjustments. It can set itself up across even very large environments and be expanded gradually over time. It comes with a built-in trial setup that demonstrates and tests its capabilities out of the box, and - as already noted - it can take over your existing utility jobs without rewriting them, so you start seeing benefits quickly with little upfront effort.

Deploying across Db2 subsystems

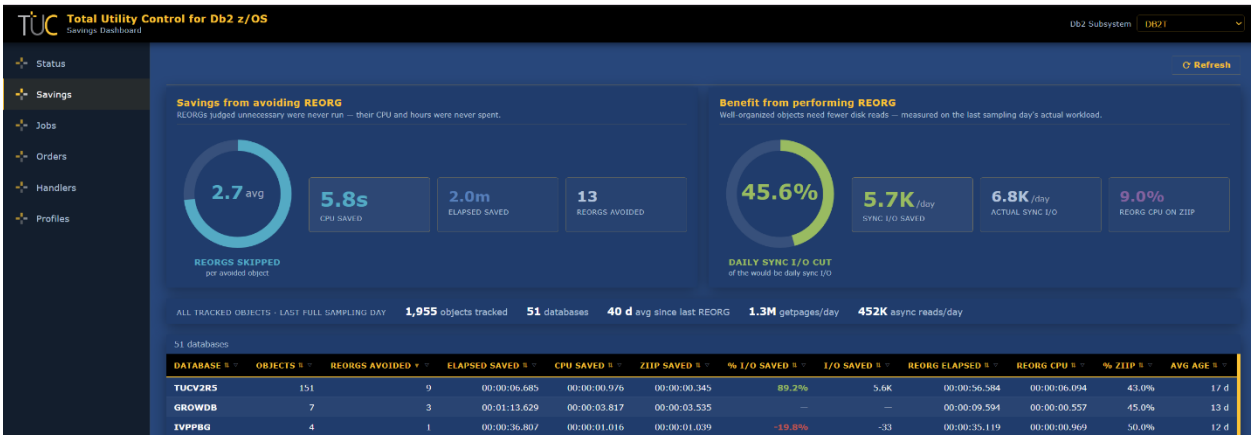
Once TUC is configured the way you want on one subsystem, you can copy that configuration to your other Db2 subsystems, compare subsystems to spot differences, and stand up a new subsystem consistently in minutes rather than rebuilding everything by hand. Running multiple subsystems the same way becomes simple and low risk.

MEASURING THE RETURN

One of TUC's strengths is that its savings are not theoretical - they are visible. From TUC's dashboard, you can see how much backup volume incremental copies save, how much statistics collection work is avoided because the data has not changed, and how much processing time is saved by not running unnecessary reorganizations. TUC tracks all this for you directly. TUC's web dashboard brings these metrics together in one place, presenting both the savings and the live progress of maintenance at a glance. Any report or alert can be emailed automatically in your choice of format - so the value is easy to monitor and to share with the DBA team and with management without anyone having to log into the system.

On top of the above measurable resource savings are the harder-to-price gains: fewer outages, faster recovery, and a database team freed from repetitive maintenance work to focus on higher-value projects. Taken together, TUC pays for itself by doing maintenance intelligently - only when it is genuinely needed.

Tracking the Numbers



What TUC Measures	What It Shows
Backup Volume Saved	How much backup volume incremental copies save versus full backups.
Statistics Work Avoided	How much statistics-collection work is avoided because underlying data has not changed.
REORG Time Saved	How much processing time is saved by avoiding unnecessary reorganizations.

ABOUT LOG-ON SOFTWARE

Log-On Software provides Total Utility Control (TUC), intelligent automation software for Db2 for z/OS environments that governs backup, reorganization, and statistics maintenance so that this work is performed only when it is genuinely needed.

TUC replaces blanket, calendar-driven maintenance schedules with policy-based decisions grounded in how much each database object has actually changed – reducing CPU consumption and manual effort while protecting recoverability and availability.

CONTACT

For more information about Total Utility Control for Db2 z/OS, visit www.log-on.com or email ask#log-on.com