Using a Persistent Data Store with System Center Orchestrator

Best Practices Guide

Kelverion

Table of Contents

Purpose

It is Kelverion' s recommendation that in nearly everycase runbook automations should be implemented utilizing a small backend SQL database table or set of tables that resides on the same SQL instance as the Orchestrator database. This is what we refer to as a Persistent Data Store (a.k.a. PDS Live).

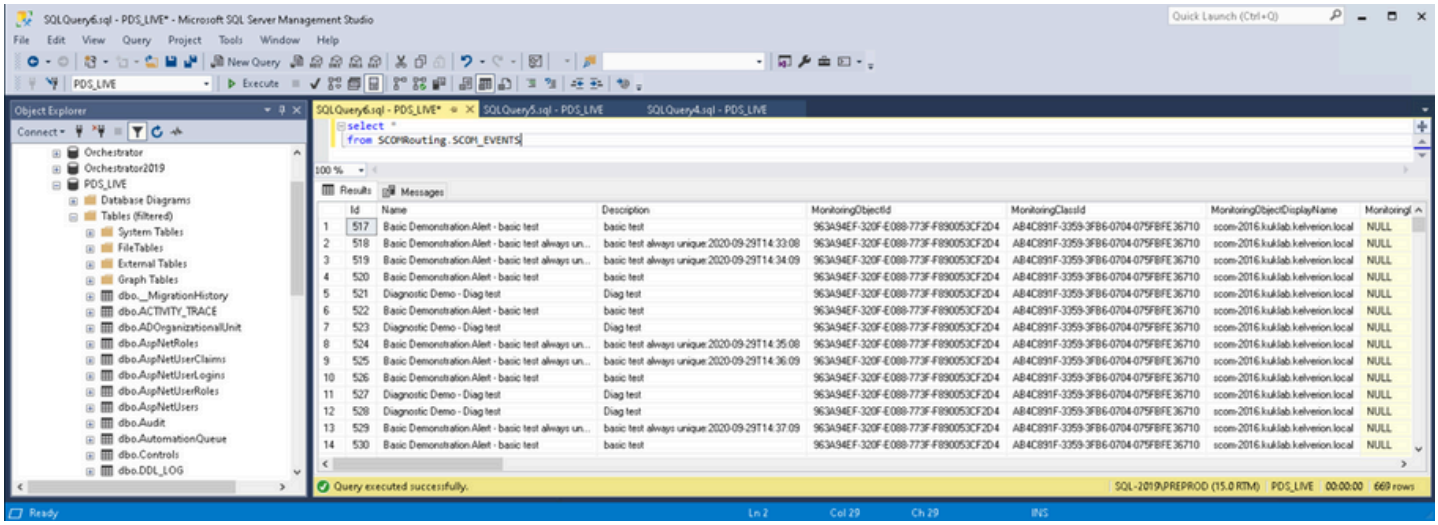
There are multiple reasons why this is a valuable practice, and we will go into detail on each one:

- Persistence
- Passing Data Efficiently Between Runbooks
- Manageability
- Re-Running and Troubleshooting Runbooks
- Waiting and Manual Intervention
- Error Handling
- Speed of New Development

Persistence

The ORCHESTRATOR database is extremely powerful, but it is not persistent which means it cannot remember or keep track of what has been done before without explicitly passing that information on to the next task. This means that if the ORCHESTRATOR runbook server were to reboot or the services were to crash, all data about executing runbooks would be lost. So, a complicated multi-step workflow that is counting on the data bus for its current state would need to be restarted manually by an administrator. That administrator would potentially need to take significant time to understand what parts of the workflow had run successfully and what parts were still needed. If a persistent data store is used at the backend, the last successful step in the process has been recorded in the database with all the necessary information to continue the next step. This provides a basic audit trail of what has been done so the administrator can easily pick up where things left off after the issue has been addressed.

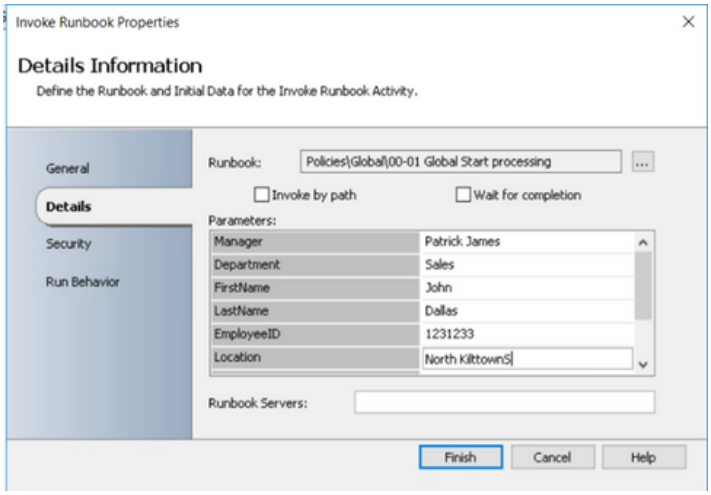
Fig 1: Kelverion's SCOM 2022 Connector PDS



Passing Data Efficiently Between Runbooks

Information gathered and generated by one runbook is not available to other runbooks in Orchestrator without implicitly passing that data through the Invoke Runbook activity. In that method, every data type needs to be defined in the destination runbook and the InvokeRunbook activity. This means passing large amounts of data between runbooks is a time-consuming and error-prone process. By writing the data to an SQL database at the end of each runbook, the data can be passed by simply supplying a secondary runbook with the primary ID of that row in the database. This also allows runbooks to act on data collected from multiple other previous activities. In the case of a step that sends out a final status email, it would only take a pull from the database to retrieve the status of all the previous steps and include their data in the notification.

Fig 2: Passing Data between Runbooks Directly



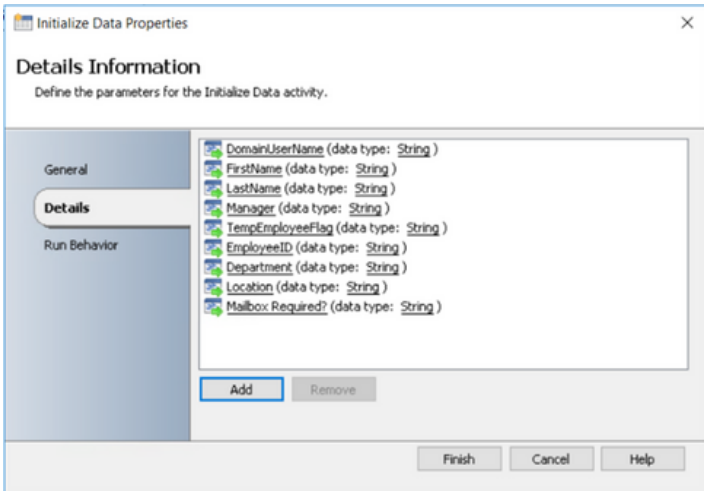
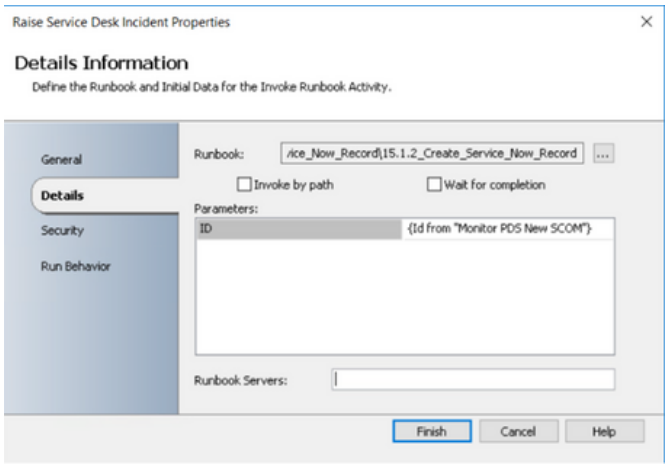
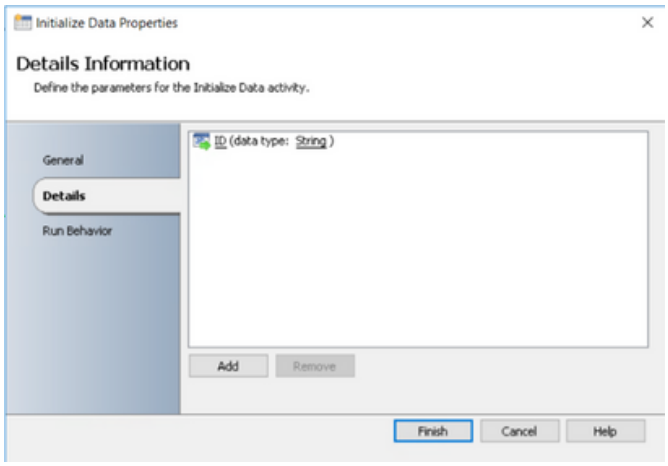


Fig 3: Passing Data Using a Persistent Database





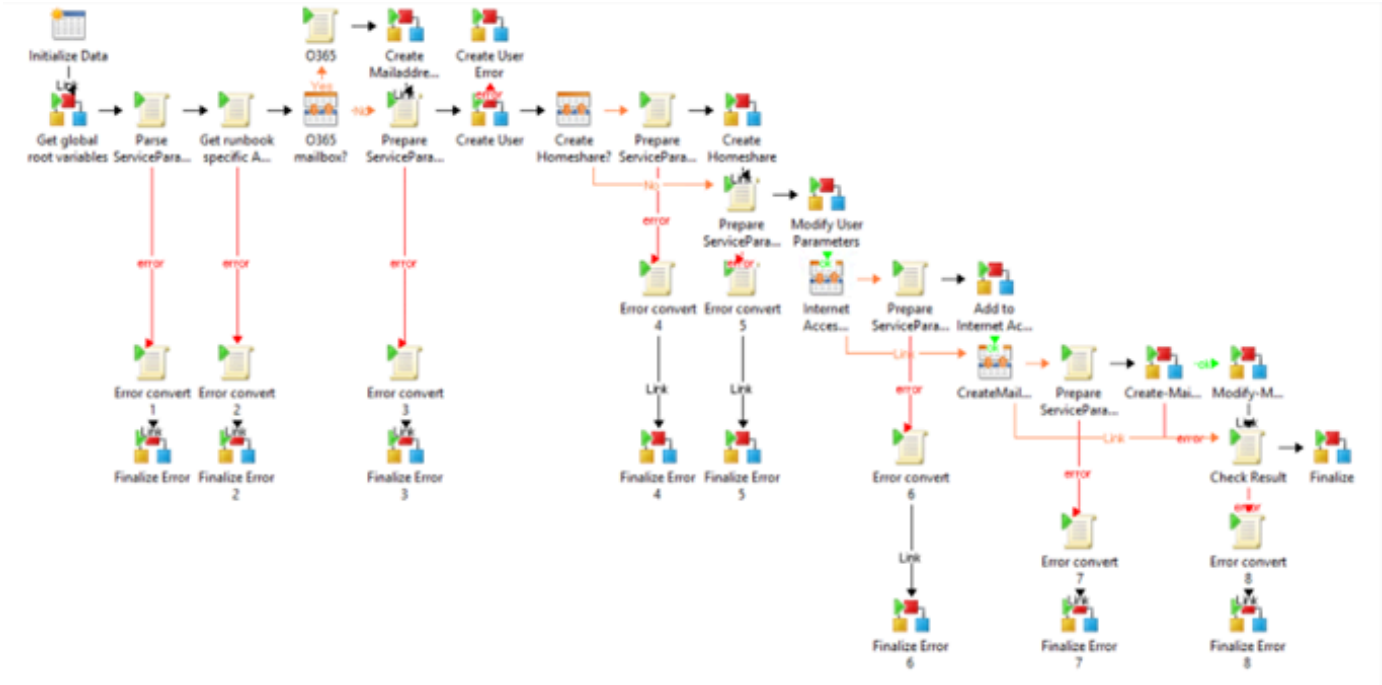
By writing the data to an SQL database at the end of each runbook, the data can be passed by simply supplying a secondary runbook with the primary ID of that row in the database. This also allows runbooks to act on data collected from multiple other previous activities.

Senior Automation Consultant, Kelverion

Manageability

Because of some of the limitations inherent in the databus, runbooks written without a database methodology tend to be large and harder to manage. This is an example of such a runbook:

Fig 4: Example Runbook without a Persistent Data Store



"Kelverion's support enabled us to implement more efficient runbooks"

System Analyst, Global Food Corporation

By using a database at the backend of a runbook process we are able to make smaller more manageable runbooks that work together in a modular fashion. These runbooks can handle smaller tasks and then pass the data reliably to the database, and then another concise runbook can pick it up from there. This makes it significantly easier to add, remove, or change an existing workflow.

Fig 5: Example Runbook using a Persistent Data Store

The positive experience that I had working with you both actually was mentioned in my statement of work presented to the management, as one of the deciding factors on why we should purchase products from Kelverion.

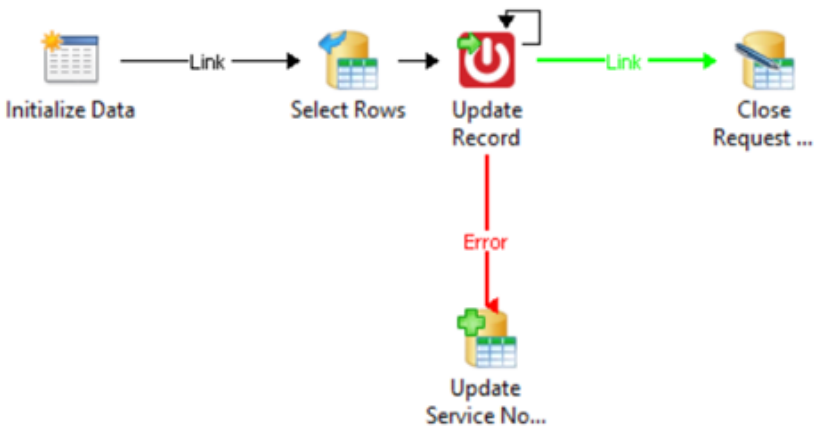
Senior Automation Analyst, Oil & Gas Company

Troubleshooting Runbooks

The use of a persistent data store makes re-running a step in the process significantly easier and in some cases (See Figure 4) even possible in cases where it would not be without one. Consider the examples given in Figure 2 and Figure 3. If we needed to re-run the process shown in Figure 2 manually, we would need to acquire all of the new user information and then type that data into the Runbook Tester. In Figure 3, where the data is stored in a database, all we would need to re-run the process is the "ID".

Figure 6 below shows how Kelverion structures many of its runbooks when using our best practice by including a persistent database. The Initialize Data activity is receiving the ID of a table row from another runbook. The data from that table is then made available on the data bus by the Select Rows activity. In this example, the Insert Record activity from Kelverion's Integration Pack for ServiceNow uses the data to create a new ticket. Kelverion's UpdatedRows activity is used to record the success or failure of the task. Additionally, the information from the Insert Record activity, like the ticket number, can be added to the data set.

Fig 6: Example Best Practice Runbook



Waiting & Manual Intervention

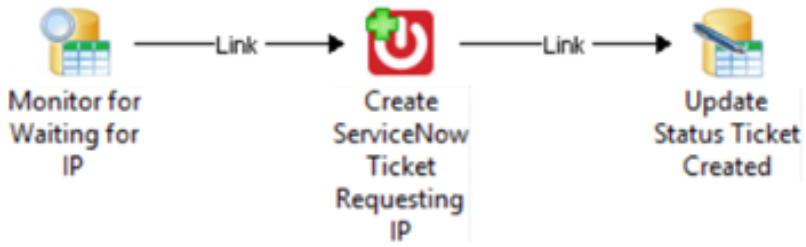
Without a way of storing information outside the data bus, the original data and any new data collected as the current runbook executes must be passed on to the next runbook immediately. Many processes gather data that needs to be used at a later time, not just the next step. Additionally, some steps in the process need to wait for an approval or manual intervention. Because all of the data needed to execute a runbook step exists in a persistent database; a step can be run reliably now or at any time in the future.

The runbook in Figure 7 starts when the filter criterion for the Monitor Rows activity has been met for a given row. Usually the monitor is looking for a status column in the table to be equal to a specific value. This runbook then uses a REST API call to request an available IP address to assign to a new server. If the IP address is provided, then it gets recorded back to the database along with a new status that will kick off the next step. If the IP address is not provided the process does not need to fail. The status of the process is set to a value that will in turn run a process that notifies the appropriate team that an IP address is needed and then once the IP address is provided the process can pick up where it left off.

Fig 7: Runbook Example with Manual Intervention Option

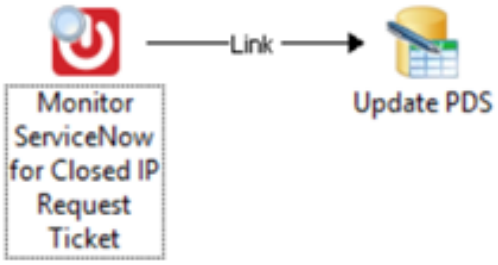
The manual intervention could be handled through the company's ticketing system. In Figure 8 the information from the PDS is used to create a ticket and request an IP Address from the correct team.

Fig 8: Create a Service Desk Ticket



A runbook would then monitor that type of ticket for closure. The closed ticket should contain the new IP address. The address will then be written into the database with an updated status. That status would be the same as if the runbook in Figure 7 had successfully found an IP. Whatever runbook would normally come next would now execute.

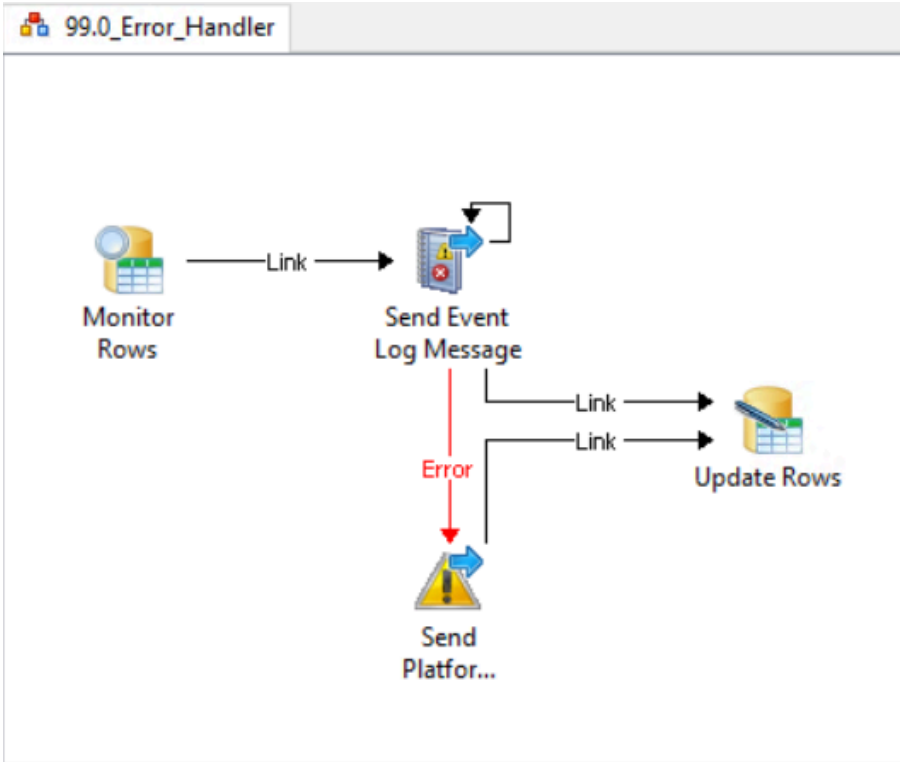
Fig 9: Monitor for Closed Ticket



Error Handling

Error handling is one of the most critical aspects of good runbook design. Trying to handle errors in each runbook would add a significant amount of work and potential rework. However, by creating an Alert Handling database table error information can be written from any runbook to this table. Administrators can be notified using a single runbook that looks for new entries in the Alert Handling table. This table would have historical error information that can help identify error trends in the workflow. Figure 10 shows an example of this approach. You should also note, this approach can be used to account for multiple possible errors received and directed on to another process or runbook to resolve the error, then return to where it left off.

Fig 10: Common Error Handler



Speed of New Development

The process of developing new runbooks can be a time-consuming process. One of the primary metrics to evaluate when deciding whether a certain automation project should be done is the amount of time it will take to build and maintain compared to the amount of time saved. Kelverion has found that using this methodology involving a persistent database with a modular approach to runbook design in combination with our Integration Pack for SQL Server reduces the overall time required to develop enterprise class workflows in Orchestrator by two thirds. If it takes substantially less time to create, maintain, and update workflows many more taskscan be automated. This results in an IT department that provides services more quickly, with less opportunity for human error thereby ensuring greater consistency, all combined for less cost and a substantial savings.

"You guys are rock stars. I always got immediate and efficient support from both of you and really enjoyed working with you on automation.

Senior Automation Analyst, Oil & Gas Company

Kelverion Integration Pack for SQL Server

The methodology described above is made most possible by the Kelverion Integration Pack for SQL Server. The methodof using a backend databasefor runbook designcan be partially accomplished with Orchestrator's built in Database Query tool but with significantly less efficiency and almost none of the time savings.

Kelverion's Integration Pack for SQL provides Orchestrator with the following activities:

There are a number of significant advantages the Kelverion Integration Pack provides over the built-in SQL tools:

- No SQL code required.
- Published data does not need to be parsed.
- Efficient communication with the SQL server.

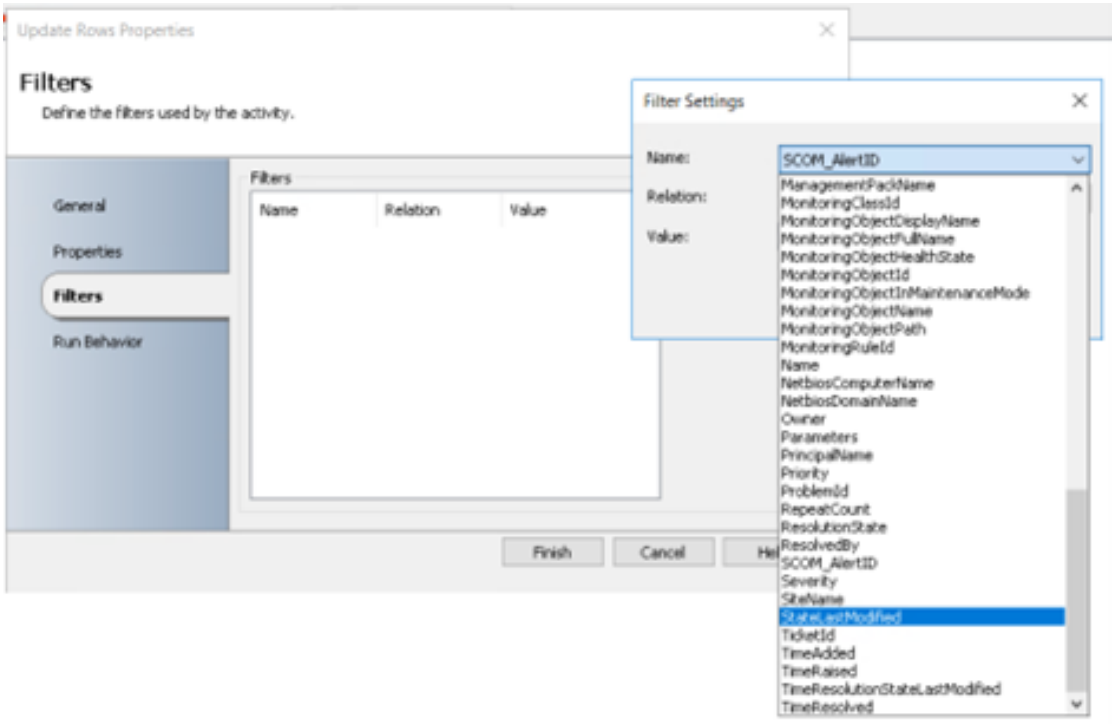
"Kelverion's integrations worked right out of the box with no Runbook server modifications, a huge point I made to my manager...Thanks for all your help!"

Technical Administrator, US Financial Services Organization

No SQL Code Required

All of the activities use a simple interface to monitor, filter, select, and update data. With the built-in tools, SQL commands will need to be written and maintained for each activity.

Fig 11. Example of Select Rows Filtering



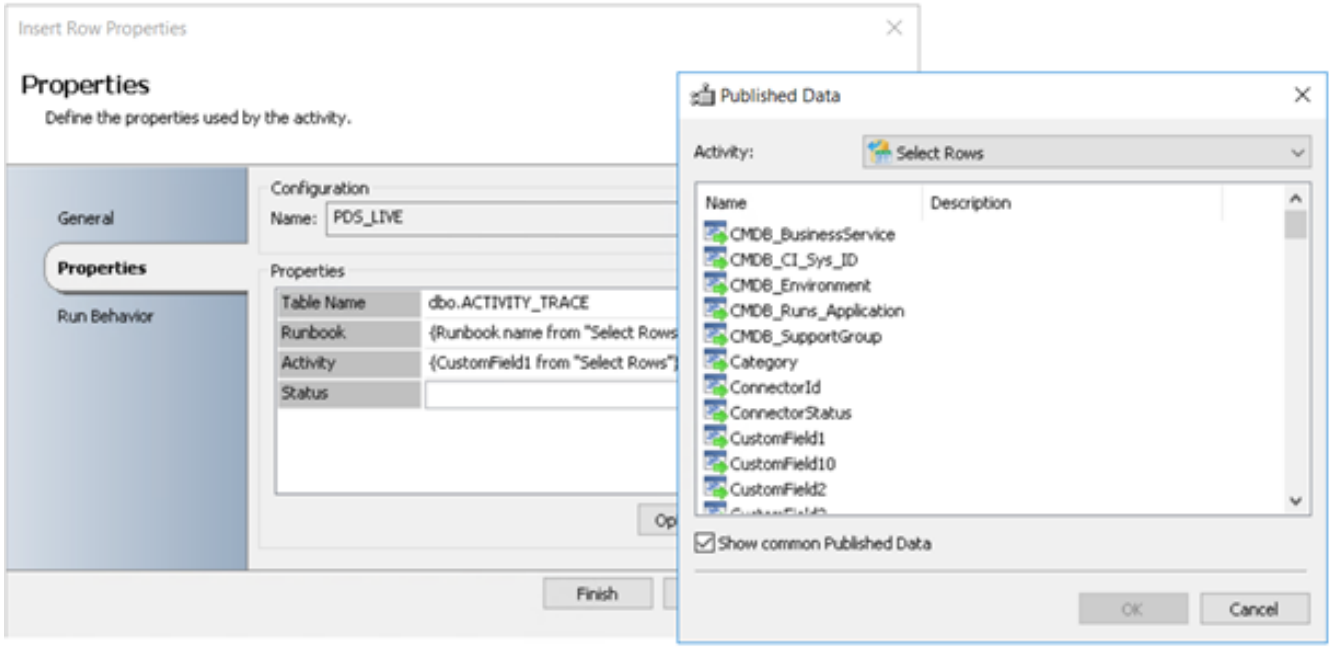
"Kelverion has a lot of integrations that we use to make automation easier in our runbooks; we have not seen this amount of integrations covered by one company or done as well."

IT Administrator, Paint and Glass Manufacturer

Published Data Does Not Need To Be Parsed

Data retrieved from SQL by the built-in tools is presented on the data bus as Comma Separated Values. This data needs to be parsed each time it is accessed before the data is actionable. Data retrieved by Kelverion's SQL activities are presented on the date bus already parsed and are represented by their associated column names.
(Note: Kelverion's Run Procedure activity returns data in a raw format.)

Fig 12. Example of Published SQL Data



Efficient Communication with SQL Server

Each time the built-in SQL activity executes, it opens a connection, authenticates, takes its action, and then drops the connection. This happens for each row. Kelverion's activities use connection pooling. Activities connecting to the same database use a single connection. That connection is only dropped if it is idle for more than 60 seconds. This significantly speeds up the execution of large multithreaded processes and reduces resource utilization on both SQL and Orchestrator servers.

These are only a few of the advantages of Kelverion's Integration Pack for SQL Server. Please find out more about the power of the IP for SQL Server and request an evaluation copy at www.kelverion.com.

Further Reading

System Center Orchestrator Best Practices

Top 10 IT automation mistakes and how to avoid them

Downloadable templates for runbook documentation

Planning an IT Automation Project

Microsoft System Center Infrastructure Monitoring and Automation in Action

Solving Common System Center Orchestrator Problems

About Kelverion

Experts in Cloud, On-Premise and Hybrid automation, Kelverion provide solutions and integrations that remove the manual process tying up IT staff; transforming the productivity, efficiency, and supportability of IT service automation. Our products utilize and enhance the power of Microsoft Azure and System Center Orchestrator.

Working closely alongside Microsoft, we have developed our integrations and automation solutions to help bridge the gap between Microsoft's automation platforms and third-party systems, in the process building key alliance partnerships with multiple vendors to ensure our products are fully certified.

Since 2010, Kelverion has expanded to become a global company, with offices now in the UK, Canada, and the US. Through this, we are able to offer and support products and professional services engagements to enterprise-level organizations no matter where they are.

There's a smarter approach to IT Service Management

Get in touch to find out more

www.kelverion.com

info@kelverion.com

US: +1 289 801 0559
UK: +44 203 875 8035

https://www.linkedin.com/company/kelverion-automation/