Self-Service Troubleshooting in TimeXtender

Best practices to solve technical support issues in less time!

view this article on web

Software solutions and frameworks simplify tasks and increase productivity. In addition to being accurate, fast, low-cost, compliant, and secure, vision for modern software ecosystem includes **self-describing** (easy to learn), **self-diagnosing** (handle anomalies), **self-healing** (recover from failures) and **self-supporting** (enable users to fix issues).

When designing and automating modern data estates with integrated subsystems and external components, some technical issues may arise in areas such as ***how to, setup, configuration, connectivity, performance, monitoring*** *and* ***best practices***. A typical TimeXtender project interacts with the host operating system, multiple machines or cloud VMs, **.NET** framework, cloud repository**, ODX** storage (for example, a **Data Lake**), a Repository on **SQL** Server, and many **data sources, providers,** and **endpoints** – auto generating **scripts,** using pipeline/ copy activity, making **API** calls and tracking **logs**.

*Does **self-service** apply to building modern data estates and data warehouse projects?*

*Is there an organized & **scientific** approach to focus and solve complex issues with less back-and-forth e-mails and chats?*

Analyzing technical support cases from partners and customers in my **Customer Success** career, here are some tips to expedite resolution. While this article's focus is on SQL databases, Azure, TimeXtender and Windows environment, similar tips may apply to other systems as well.

Is the Solution already documented somewhere?

It is likely that someone has experienced these issues before. Instead of re-inventing the wheel, we can find best practices already documented. For example:

- Lookup support site for Knowledge Base, User Guide, and community forums for known issues and tips – check to see if another user has discovered a similar design pattern before.
- Review online training for "*How To*" tutorials.
- When working with **REST APIs, JSON**, **XML** and other no-SQL data sources, look for specific guides covering methods to connect and extract data.

Clearly define the problem for Support:

Include all the necessary information for the support team to expedite research. Documenting and deep thinking over a technical issue often reveal fresh ideas, workarounds and solutions.

- **Environment** - how many machines or endpoints (on-premises or cloud) are involved in this overall scenario? An **architecture** diagram helps summarize a complex environment. *A picture is worth a thousand words!*
- Business **impact** and deadlines (if any)
- **Software versions** (including databases, providers and components). Add an image of **Help ->** "**About"** dialogue.
- **Detailed reproduction steps** (simplified steps to re-create the problem)
- **Relevant error messages** or **log files** (copy-paste error messages in **text,** where possible)
- **Screen images** - **environment, settings,** and **configuration options** used in current setup
- **Frequency**: is this an intermittent/ transient (rare occurrence, hard to reproduce) or consistent issue? Was it working before, or did this only occur in a new implementation or after upgrade?
- When researching **design** best practices, or unexpected behavior among hundreds of objects and relations in a Data Warehouse, consider making a **video** or screenshots showing current state and UI operations.

Consider potential workarounds:

Is there an **alternate or better way** to implement it with TimeXtender (with less time and resources)? *Example: If a provider's remote connection is slow, we can upload (or download) source data to a staging server and then use a faster method to extract, or use a better connector?*

As we implement data flow from source to ODX, then to a Data Warehouse and towards endpoints - there are various configuration **options** and **settings** to tune and optimize. Explore the following choices:

- an alternate data source **provider** (custom provider, ADO, OLE DB, Azure Data Factory, CData)
- an alternate **data movement** method (for example, ADO vs. Azure Data Factory)
- **timeout** settings at various stages to process large data transfer
- increase/ decrease **concurrency** (multiple threads) to fix a performance, racing or deadlock issue
- **scale up** cloud resources to solve a performance, latency or size issue
- *batching, chunking, incremental load, or custom query* – review the best **strategy** for your use case

- *data type* ***conversion****, cast or override* – find the best way to load and process data from a variety of sources
- enable row and page-based **compression** on larger tables to reduce size and improve performance

Sandbox - test environment:

You may setup a separate sandbox environment with the same software version as in production, and let it run in parallel for testing. This may help **repro, prototype, analyze** various troubleshooting methods, workarounds, and fallback plans without affecting the production system.

Is the problem already resolved in a newer version?

While an immediate upgrade is not always feasible, check to see if a newer release includes the feature or fix you are looking for.

- Review the latest **release notes** and Release Documentation**.**
- Upgrade in the "**sandbox**" and apply acceptance tests without changing the production system first.

Isolate environment and operating system issues:

Modern software systems operate in an eco-system of host environment and external tools. For further troubleshooting:

- Lookup community forums for known issues and explanations on **error messages**, generic *HResults*, or "*unknown*" exceptions. This is useful if the underlying framework returns an error code without a descriptive message.
- Use Windows **Event Viewer** to identify issues during startup, execution, sync and transfer
- Use Task Manager, Process Explorer, Process Monitor for insight on CPU/ memory usage. View all TimeXtender and ODX related **helper processes** and other services.
- For strange errors occurring on a specific machine only, run sfc (System File Checker)

Isolate *Data Source* specific issues:

When the problem occurs in extracting data, use connectivity and tracing tools for that specific source. For example:

- Use a profiler or query tool for SQL Server and other DBMS. *Example: SqlLiteStudio can view a backlog file*
- Enable **tracing** for **ODBC** data sources. Try switching from 32-bit to 64-bit, or from "User DSN" to "**System DSN**" to see if that enables data extraction
- Enable **logging** for CData providers
- Try an alternate form of **connection string.** Review provider docs and ConnectionStrings.com
- View **Data Lake** folders and files via **Azure Storage Explorer**
    - View **.parquet** files in a visual tool to examine data being transferred
- Compare generated scripts and logs for "**working**" and "**non-working**" scenarios
    - Use a graphical tool with syntax highlighting to compare side-by-side

For root-cause, narrow down and analyze:

It is time-consuming to troubleshoot an issue with large systems and many moving objects. Often, we can quickly solve the same problem by splitting it into smaller parts, or by removing ambiguous layers (divide and conquer).

- Eliminate variables, **simplify** the repro, and isolate the cause. *(Example: does **retry** or **manual** execution resolve the issue?)*
- **Connectivity/ authentication/ permissions**: use an external tool such as SSMS (SQL Server Management Studio) or Power BI to verify the same credentials
- Ensure TimeXtender **Scheduler** Service is running when you **schedule** a package to execute.
- For **cloud** resources, ensure the right **tenant** and **key** are used, and **application role** is configured.
- Visualize and trace **end-to-end data movement** of specific row(s) through various layers and subsystems, from the source to the endpoint. This may help troubleshoot **history**, incremental load, data type override and similar issues.

Upgrade checklist:

Agile development, emerging technologies and feature improvements lead to new releases periodically. Before upgrade, save and close current projects and ensure any scheduled package executions are not running on any environment. Upgrade the ODX first, then TimeXtender. Add the following post-upgrade verification:

- Review new release documentation for **changes** in architecture, data flow, settings, or tracking (logs)
- Ensure the software is configured properly and **existing projects** are building OK.
- Ensure the previous version's Windows **services** are stopped or disabled, and newer version's Windows services are up and running.
- Upgrade both the ODX and TimeXtender to the latest version where possible.
- Verify that **existing data connections** refresh, load new providers, and operate as expected
- Run the **Performance Recommendations** tool

Proactive maintenance, cleanup & backup:

Over time, the underlying log files for SQL databases may grow, Repository database and ODX backlog may also grow. This may cause slower response for certain tasks.

- Work with your DB administrator to automate a process for **backups, log shrinking** and Repository **cleanups**
- Periodically **backup** your **Repository** database which contains all your projects and their versions.
- Remember to make a valid backup of your **repository** before running any Delete queries! (*Do not run Delete queries in Repository or backlog unless TimeXtender Support gives specific instructions*)
- Write descriptive notes for **project versions** after reaching major milestones in design. This may help restore or **revert-back** to a working configuration when needed.
- Upon project completion (or reaching an important milestone), **export** your project as an **XML** file and generate its **documentation**.
- Upgrade your host OS, apply the latest **service updates** for SQL Server, ensure an appropriate version of **.NET** framework and **Java** runtime are installed.
- Uninstall old versions of TimeXtender not in use.
- Clean up **ODX storage**'s leftover folders from unused data sources. Each full load adds a new version of data to storage. Use **Storage Management** tasks to delete or archive old Data Lake folders into "**cool**" storage.
- Run SQL **Database Cleanup** wizard to identify old/ unused objects which may be dropped to save space.
- When using Azure **Marketplace** template, keep a backup of **VM image** for future migration and upgrade.

- When designing and testing a large project over a period, it is a good idea to save and close the project periodically (to **release memory** resources).

Networking issues?

Modern software such as TimeXtender may communicate with remote machines and cloud resources for meta-data access, API calls, data movement between user configured databases, project deployment from *Development* environment to *Test* (or *Production*).

- Review **ports** and communications between servers and machines involved
- Enable or **white-list** specific addresses or ports as appropriate.
- Use specific *servername* instead of 'localhost' or '.'
- For server "remote name could not be resolved", review if there is any change on DNS Servers
- For scenarios related to **latency, throughput,** or **congestion**, use a packet sniffer or **network debugging** tool such as **Wireshark**. For dedicated **HTTP/HTTPS debugging**, use **Fiddler**.

Antivirus impact on running processes.

During an ETL pipeline - ODX sync and transfer tasks, Data Factory pipeline activity, manual Deploy and Execute as well as scheduled package execution - TimeXtender, ODX and Self-Hosted Integration Runtime (if in use) may launch several **helper processes** (visible in Task Manager). It is important that all those processes and corresponding windows services are running smoothly without interference from external factors, server maintenance/ reboots and Antivirus tools.

- We recommend adding TimeXtender and ODX related applications to your **antivirus "whitelist"**

Cloud intermittent issues?

Running a data access framework in the cloud for a period, we may come across transient issues which resolve quickly. For example, a brief connectivity error could occur when a system reconfiguration is in progress.

- Check Azure **Service Health**
- View Azure resource **monitoring** tools and metrics
- View Data Factory **pipeline** activity
- Implement **retry logic** in execution package
- For Azure Data Factory connecting to on-Premises data sources, ensure the **Self-Hosted Integration Runtime** is installed and running.
- Check for resource usage and **scaling** options

Cloud cost issues:

To save cost on cloud deployments, configure and *right-size* external resources:

- Before Deployment - use Azure **Pricing Calculator** to **estimate** the total cost of all resources
- After Deployment - view **Cost Management** options in Azure Portal. Set **budgets**, **thresholds,** and **alerts**
- Review Azure **Advisor** recommendations and **cost-saving tips** based on your usage patterns
- **Pause** Azure resources (Azure Synapse, Analysis Services etc.) not in use
- Use **serverless** option where available
- Identify **unused** resources and VMs. Stop and deallocate VMs not in use
- Clean up old Data Lake folders from **unused** data sources – delete or archive into "**cool**" storage.

We hope this article and comprehensive checklists help data architects, data engineers and data scientists design and maintain big data framework, extract from any data source and build modern data estates for BI and AI in less time.

Creative troubleshooting is a combination of verification processes, checklists, quick debug tools, know-how of hybrid/ external parts of the puzzle, and applying proven practices. Review our article's section headings again to make your own top-10 tips for self-support.

*Takeaway lesson: Eliminate variables, **simplify** the repro, and isolate the root cause!*

Happy Troubleshooting! with Data, Mind and Heart. Because Time Matters.

view this article on web