
DataFlux dfPowerStudio Quick Tips, Tricks, & Best Practices



A NEOS Whitepaper

Author:

Brian Snyder, Architect, Information Architecture Services, NEOS

Date:

February 26, 2009

Overview

DataFlux dfPowerStudio is a suite of products designed to identify and correct data quality issues in virtually any data source. While teaching the dfPowerStudio introductory class, very often students want to know not just *how* to use dfPowerStudio, but how they *should* be using dfPowerStudio to build the most efficient, maintainable, and accurate processes possible. The answer to that question can be broken into 3 categories: setup/configuration, design, and process. This document focuses specifically on the first 2. The content of this document is the result of several years of experience designing and building production dfPowerStudio processes for clients, as well as teaching the dfPowerStudio class. Much of the information listed below comes from personal trials and tribulations, as well as advice from some truly exceptional people at DataFlux.

This document is intended as a “Quick Reference” of Tips, Tricks & Best Practices. It is not intended as an in-depth examination of Best Practices for DataFlux. Follow-up documents will offer more in-depth, focused perspectives on various aspects of using dfPowerStudio. Further, this document primarily focuses on the *mechanics* of configuring and using dfPowerStudio, as opposed to the methodology of designing and building Data Quality processes.

This document is intended as a “living document”. As such, additions and modifications will be made regularly. Data Quality tools are evolving at an exciting pace. As the premiere software vendor in the Data Quality space, DataFlux is constantly updating and improving its tools. As these updates and improvements are made, this paper will be updated to include the newest features and methods. Stay tuned at our website, <http://www.neosllc.com>, for updates to this document and future white papers.

Audience

The DataFlux Tips, Tricks, & Best Practices document is intended for those who have either completed the DataFlux dfPowerStudio training, or have experience using DataFlux dfPowerStudio to build Data Quality jobs. This document is not intended as an introduction to dfPowerStudio or its use.

Document Structure

The tips, tricks, and best practices have been broken up into a group of categories. These categories align either with a functional component of DataFlux, or a methodology space such as “Migration / Source Control”. Each section begins with a brief description of the subject area, followed by a bulleted list of tips, tricks & best practices.

About the Author

Mr. Brian Snyder is an Information Architect within the Data Management practice at NEOS, a professional services company focusing on enterprise data, application and modernization solutions for international companies. Mr. Snyder is an accomplished technical architect with over 15 years of database design and implementation, data analytics and data management experience. Mr. Snyder has worked with some of the worlds most recognizable organizations in implementing data quality and management programs and solutions. He has been an Information Architect with NEOS LLC in Manchester, Connecticut, for since leaving The Hartford Financial Services Company in 2006 where he was lead architect for their Enterprise Data Services division. Prior to The Hartford, Mr. Snyder was in various lead architectural roles with several large companies.

Quality Knowledge Base (QKB)

The Quality Knowledge Base (normally referred to as simply the “QKB”), is a set of files which contains the Data Type definitions that DataFlux uses in order to parse, standardize, match and process data. As such, the information in the QKB is **critical** to most data quality flows built in DataFlux. The QKB has a version number which represents the year and release of the QKB (ex. 2008A). Historically, DataFlux has released 1-2 new QKB versions per year. With each new release of the QKB, DataFlux continually improves the performance of existing Data Types (improved parsing, standardization, matching, etc) and adds new data types.

Because of the importance of the QKB, the related tips top this document:

- Always use the most recent version of the QKB.
- All dfPowerStudio installations (each PC) and the DIS (each server) should be on the same version of the QKB. If a job is run on a different version of QKB, you may not get the same results.
- If you store match codes in a database, the match codes must all be regenerated when you upgrade to a new version of the QKB.
- Any modifications or additions to QKB files need to be propagated to all dfPowerStudio installations and DIS servers. *(Example: if you create a scheme and use it in an Architect job, the scheme file must be copied to the DIS and other dfPowerStudio installations before the job can run there).*
- If you create a standardization scheme in dfPowerStudio, it can be found in the following directory (default): C:\Program Files\DataFlux\QltyKB\C\2008A\scheme (2008A is the version, this will change if using a different version). The file name should be the same as specified when saving the scheme. A scheme is made up of 1 file, with a “.sch.qkb” file extension.
- When creating a standardization scheme, develop and use a scheme naming standard. I recommend developing a standard prefix to differentiate all custom schemes from built in schemes. I recommend using a 3 to 4 character prefix indicating your organization’s name. Keep the name as short as possible, so it will be easily visible in the drop down in Architect.
- If you own dfPower Customize and need to modify an existing Data Type, I strongly recommend that you make a copy and modify the copy. As with schemes, name any new or modified data type definitions with a prefix to make your custom types easily identifiable.

Macro Variables

DataFlux jobs will often be built, deployed and executed in a variety of environments. Most organizations today follow the best practice of maintaining information systems with multiple environments which align with the software development lifecycle (dev, qa, production). Certain types of information required for a job to run will vary from one environment to another (Database connections, file output locations, etc). In order to handle this type of variability, DataFlux offers macro variables. Macro variable essentially act as a placeholder within a DataFlux job, which will be replaced at run-time with the value defined within the specific environment. Proper use of macro variables is a critical component of developing maintainable jobs which will seamlessly move from one environment to another.

- Macro variables should always be used for the following items:
 - ODBC DSN for Database related nodes (Data Source, SQL Query, Data Target Insert, Data Target Update, SQL Lookup, SQL Execute, etc)
 - Repository Name (Specified in Data Monitoring Node)
 - Paths for File input/output nodes (Text File Input, Text File Output, etc)
 - Path for all Embedded Job Nodes (EJs)
- Macro Variables should only be set in the **Advanced Properties** of Architect job nodes. To access advanced properties, right click on any node in an Architect job and select “Advanced...”. If a macro variable is set in the standard properties page, it will immediately be de-referenced, losing the variable nature of the macro.

- Macro variables should be referenced preceded with and followed by "%%". For example, a macro called "INPUT_DIR_PATH", would be referenced as: %%INPUT_DIR_PATH%% in the architect job. Please note that when creating macros, you do NOT use the %%'s.
- Macro Variable definitions are stored in the "architect.cfg" file, which on the dfPowerStudio client is stored by default in: c:\Program Files\DataFlux\dfPower Studio\8.1\etc. The architect.cfg file contains macros in the following format: <MACRO NAME> = <VALUE>.
- Macro Variables can be set in the following ways:
 - The "Macros" Node in the dfPower Studio Home screen. New macros are created by clicking the "+" icon.
 - Directly editing the architect.cfg file.
 - In an Architect job, by selecting "Tools->Options->Global Values" from the menu. *** Please note that any macro variable set here only persists for the current Architect session. This does NOT add the macro to the architect.cfg file.*
- If you define a new macro variable while Architect is open (except for temporary macros set in "Tools->Options->Global Values"), you must close Architect and re-open it before the new macro variable is visible to the job. *Macro Variables are evaluated when Architect starts.*
- In order to share jobs that use macro variables OR to execute a job with macro variables on a DIS, the macro variables MUST exist in the target environment.

Maximize Performance

Over the course of consulting engagements and teaching, one of the most common questions users have is, "How do I make my jobs run faster?" The performance of DataFlux data quality processes, much like performance in other technologies, is impacted by a number of different factors including: job design, optimization of interaction with external data sources, storage system performance, memory allocation, and network performance. Many of these factors are beyond the control of the average DataFlux developer (network performance, storage system performance).

What is covered in this section are relatively simple, straightforward things you can do to get the best performance out of your jobs by adjusting the things that you have the most control over: design and configuration. Much of what is covered here is specific to job design. Specific configuration items which can be tweaked for performance are found in the DataFlux Integration Server section.

As with any component that access information systems, the slowest items are generally those that access external resources such as files or databases. As such, please pay particular attention to the way that you access those external resources – several of the guidelines below speak to specific things you can do to maximize the performance of such access.

- Keep the minimum number of columns in your Architect job flow.
 - Only select the columns you need from a data source. Do not use a "SELECT * FROM".
 - Set your default Output Fields to "Target" in Architect Options (In Architect, Tools->Options -> General Tab: Output Fields).
 - Use a Field Layout node when necessary in an Architect job to remove calculated columns and other columns which are no longer required.
 - When declaring temporary variables in expression nodes (for things such as calculations), declare the variable as "HIDDEN" so the variable will not be added as column in your stream.
- When performing an Address Verification, sort the data set by Zip/Postal code, in ascending order just prior to the Address Verification Node. (Hint: Use the "Data Sorting" Node, found in Utilities).
- Perform as much work in the database as possible (SQL):
 - Avoid the "Data Source" input node. Always use a "SQL Query" node when possible.
 - If you require your data set be sorted, sort the data set using an "ORDER BY" clause in your SQL statement when possible.
 - When joining tables from the same SQL database, handle the join in the SQL statement instead of using the Data Joining node. Use of the data joining node requires all data be returned to and sorted by DataFlux.

- If your job needs to reference a lookup table to validate referential integrity, use an outer join to join the data in initially if possible. Using the SQL Lookup node will submit a SQL statement to the database for every record in your stream.
- **Only use an outer join when absolutely necessary.** I see many people universally using outer joins when an inner join will suffice. This applies to both joins in SQL statements and using the DataFlux Data Join node.
- For updates & deletes to SQL Databases (ORACLE, DB2, SQLServer), use of a stored procedure or a simple SQL statement through the Parameterized SQL node will perform better than the Update Target Node.
- **Whether performing an update/delete using the respective DataFlux node, or through a SQL statement, ALWAYS MAKE SURE THE TABLE HAS AN INDEX ON THE KEY FIELD!** For example, if executing the following: "UPDATE CUSTOMER SET MODIFY_DATE = SYSDATE WHERE CUSTOMER_ID = 1234", it is critical that the "CUSTOMER_ID" field is indexed. If not, performance will suffer greatly.
- Keep the QKB and reference data sources (USPS, Canada Post, etc) installed on a local hard drive (C drive).
- Always set a commit point on nodes which update the database (Data Target Update, Data Target Insert, Data Target Delete). Commit point can be set by clicking the "Options" button the property page. I recommend a value of 1000.

DataFlux Integration Server (DIS)

DataFlux Integration Server (usually referred to as "DIS") is DataFlux's distributed platform which allows you to execute DataFlux processing and Data Quality logic in a server environment. The DIS is a particularly powerful tool when deploying production batch jobs or production real-time services (Enterprise Edition Only), which allows DataFlux Data Quality processes to be shared with other applications and environments.

Many people keep the default "out of the box" configuration for the DIS. There are certain key elements of the DIS configuration which should be given particular attention. These attributes can have a substantial impact on performance the maintainability of the DIS environment. Below, items which you should be familiar with are highlighted and defined. In addition, a subset of configuration parameters which have a particular impact in performance are listed, along with recommendations for values.

- Understand the location and use of configuration files:
 - **DIS Config file – dfexec.cfg.** Default Location: c:\Program Files\DataFlux\DIS\8.1\etc. File contains configuration parameters for the following :
 - Location for the QKB and all reference data sources (USPS Postal, Canada Post, GeoCoding, etc).
 - Networking/Security Parameters for the DIS (TCP Port, Security Enabled, etc).
 - Server Memory settings (Sort Chunk, Cluster Memory, Verify Cache)
 - Locations of other config files (architect.cfg, profrepos.cfg)
 - Server Deployment directory paths (Architect jobs, profile jobs, real time services, log directories)
 - **Unified Repository Configuration File – profrepos.cfg.** This file contains a listing of all Unified Repositories and which ODBC DSN they use. This is a simple configuration file and can be copied from a fully configured workstation to the server – it should be identical. Default Location: c:\Program Files\DataFlux\DIS\8.1\etc. Current Server Location: c:\DIS\Config
 - **Architect.cfg** – This file contains Macro Variable definitions. The format of this file is the same as the client, however, each environment's file should be maintained separately. The value of each macro will vary by environment. Default Location: c:\Program Files\DataFlux\DIS\8.1\etc. Current Server Location: c:\DIS\Config
- For maximum performance of DIS the following parameters should be reviewed :
 - **Verify Cache** – This is a percent (0-100), which specifies how much of the USPS Postal INDEXES are cached in memory. *For all but the most memory constrained servers, this should be set to 100.*
 - **Sort chunk** – Amount of RAM (in MB) to use for an individual sort operation. Default is 128MB. I recommend boosting this based on available memory, to at least 512MB. If memory is available, 1024MB may be appropriate.
 - **Cluster memory** – Amount of RAM to use for a clustering (de-duplication) operation. Default is set to 64MB, I recommend boosting this as well – at least 512MB – more if memory permits.

- **Verify preload** – USPS Postal Database State's which are preloaded into memory. This should be set to either a list of states (MN IA IL MI) or the keyword "ALL". Normally, ALL is recommended if memory permits (takes about 1Gb of memory), however, the value of ALL caused our server to fail. I recommend sticking with a list of states for now.
- **Dfsvc max num** – Maximum number of real-time service job instances which can be loaded into memory at once. Default is 10. I recommend increasing this number – based on the number of real time services installed. At least 20.
- **Dfsvc preload [all]** – The number of instances of a real time service which are preloaded at server start. Should be set to an integer value. For example, if we specify "dfsvc preload all = 3", 3 copies of each deployed realtime service will be preloaded into memory at server start (up to max number of loaded services, per previous config parm). You can also specify a value for each service: "dfsvc preload = 3:RT_MY_SERVICE 2:RT_MY_OTHER_SERVICE" I recommend sticking with "dfsvc preload all = 3" and not setting each service individually. If your volume of requests is extremely high, you might choose to increase the number from 3.
- **** ODBC DSN / Macros – All ODBC DSN's used in jobs as well as macro variables need to exist on the DIS and be named identically!!!!**
- Default Server Listen port is 21036.

Migration / Source Control

As with any Information System artifacts which change over time, it is a well established best practice to keep such things in source control. Source control handles conflicts if more than one person edits a file at a time, it groups together files into a unit for release, and it allows you to retrieve historical versions of files in case of error. Many files within the DataFlux ecosystem are good candidates for source code control and are listed below.

- Items which should be in source control:
 - QKB Files (Includes standardization schemes)
 - Job Files (Using management resource directory format):
 - Architect Job Files (.dmc)
 - Profile Job Files (.pfi)
 - Server (DIS) Configuration Files (Separate for each env – dev, qa, prod, etc)
 - Dfexec.cfg
 - Profrepos.cfg
 - Architect.cfg
- Use management resources directory!
- Migration between environments
 - Architect & Profile Job files can be uploaded to each environment manually using the Integration Server Manager Application OR the files can be copied via the file system from one server to another.
 - Business Rules and associated constructs should be imported using the "Business Rule Manager" application.

Technical Support / Updates / Documentation

- For Expression Language programming, the DataFlux expression Language reference is an excellent resource. It can be found from your Start Menu --> All Programs → DataFlux Power Studio 8.1 → Help → Expression Language Reference Guide. It is in PDF Format.
- The DIS User's guide can be found on the server under Start → All Programs → DataFlux Integration Server 8.1 → User's Guide. I recommend that you copy the .pdf file to your local machine for reference.
- Check the DataFlux portal at <http://www.dataflux.com> frequently for updates – particularly for the postal / geo reference databases and the QKB.
- Technical support requests can be logged via web at: <http://www.dataflux.com/Resources/DataFlux-Resources/Customer-Care-Portal/Technical-Support.aspx> Tech support usually responds within 24 hours.

