Global Knowledge®

Expert Reference Series of White Papers

A Guide to Simplifying InfoSphere DataStage Change Management

A Guide to Simplifying InfoSphere DataStage Change Management

Greg Arkus, Global Knowledge Instructor

Introduction

The purpose of this white paper is to provide some guidance for simplifying the DataStage change management process while demystifying what is required to promote DataStage objects from one environment to another, or in the worst case, disaster recovery. DataStage project organization and design methodologies will also be discussed. The actual nuts and bolts of how the tools function are contained in courses and product documentation.

DataStage change management—the management and movement of programs, scripts, binaries, stored procedures, shared objects, environment settings, etc.—is nothing new and is essential to all software processes. Through a lack of understanding, managing the change of DataStage components can often become a test of wills. Picture fitting a square peg in a round hole, and you might start to get the idea. The complexity of what is required to promote components from one environment to another during the development, testing, and eventual implementation in the production environments, should be and is often a controlled activity. The old saying, "Without control there is little need for performance," applies here.

The responsibility of promoting upgraded or new software components is often accomplished by teams that have this specific responsibility. Part of the purpose for having a change management engineer or team is to enforce standards and make development teams think early on about how to simplify the process of migration and configuration, and in the worst case, rollback. Although developers often play a role, the goal is for the process to be repeatable by others in the organization. Developers should not be involved in this process unless troubleshooting is required. Lack of understanding or knowledge about the components and the breadth of a DataStage installation stretch the abilities of many change management models.

The audience for this paper is anyone involved in the development, administration, and change management of the DataStage environment and programs.

The DataStage Import and Export Process

Overview

Historically the DataStage import and export happened via one mechanism, the DataStage client, where import or export was most often interactive. Prior to Version 8.0, the first IBM release, import and export occurred in the DataStage Manager client; today it is the DataStage Designer client. Many thought this odd since DataStage is a server-centric application. The engine sits on a big computer somewhere, often UNIX, sometimes Windows, and now increasingly on Linux. Development is done on a Windows client that communicates with the engine. Most client management teams weren't fond of this if they were administering an instance other than Windows. And the fact that this was interactive was of great concern. A command-line utility was developed. While better, it was still on the Windows client. This endured for quite some time despite constant complaints by consultants and administrators about the shortcoming.

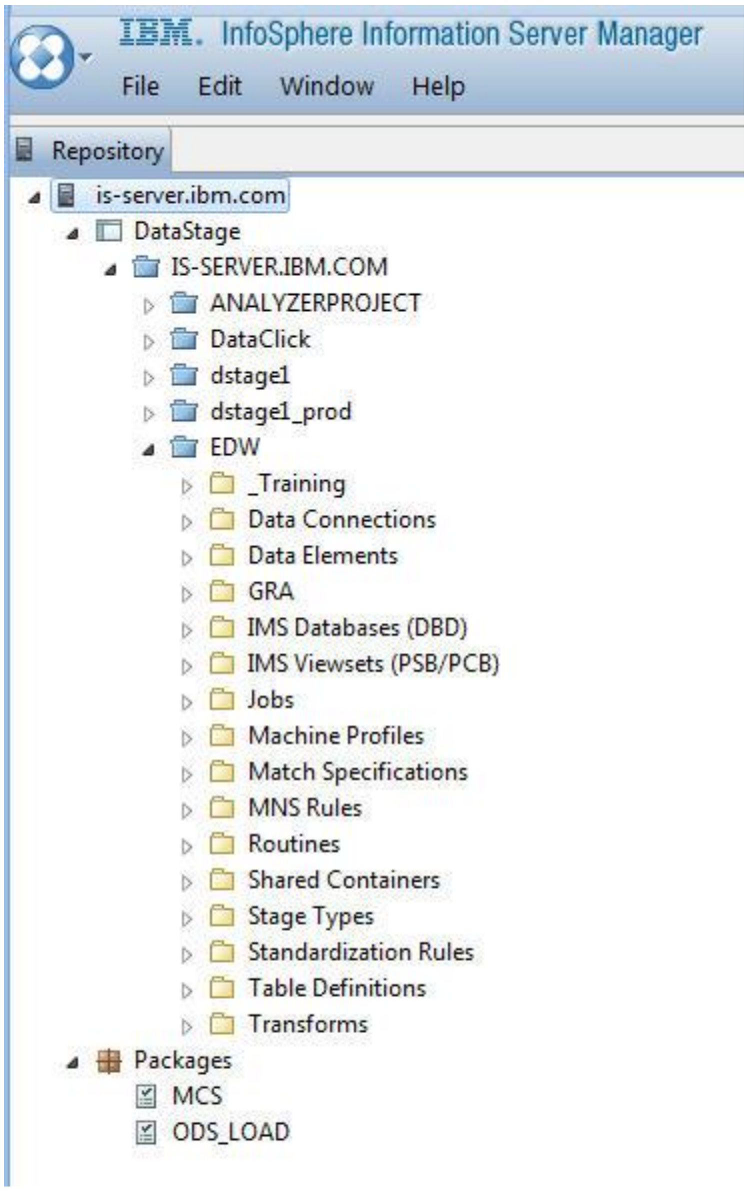
A new and improved service called the **DSXImportService** now allows you to script an import on the server regardless of platform. So, things are beginning to move in the right direction, which has made people familiar with this service much happier. In summary, a completed DataStage component(s) can be exported to a file, typically known as a dsx file. This file is a snapshot of the component that can be promoted to another project on the same or other server. It is not part of any source code control system, but it can be imported into a source code control system of your choice since it is a text file. This doesn't give you check-in and checkout ability from the development environment, but at least files are safe from change. Version control with check-in and checkout ability has not always been available to the DataStage development environment, but it is now with the support of Eclipse workspace-based source code control systems such as ClearCase or Rational Team Concert. These are used with the Information Server Manager with source code control integration on the developers' workstation.

Import and Export

With the exception of the DSXImport Service, everything is still done in one way or another on the client. Note there currently is no DSXExport Service. In summary from the client:

Export

1. Interactively from the Designer.
2. Command Line—a script can be written to select individual components or entire projects. There is an append feature. I typically schedule a script that runs nightly that exports everything. This gives a full snapshot of the code on the server, but saves it to another location.
3. As of v9.1, a versioned package/build can be created via the **Information Server Manager**. It can also be deployed to the target environment with this tool.



- is-server.ibm.com
  - DataStage
    - IS-SERVER.IBM.COM
      - ANALYZERPROJECT
      - DataClick
      - dstage1
      - dstage1_prod
      - EDW
        - _Training
        - Data Connections
        - Data Elements
        - GRA
        - IMS Databases (DBD)
        - IMS Viewsets (PSB/PCB)
        - Jobs
        - Machine Profiles
        - Match Specifications
        - MNS Rules
        - Routines
        - Shared Containers
        - Stage Types
        - Standardization Rules
        - Table Definitions
        - Transforms
  - Packages
    - MCS
    - ODS_LOAD

Import

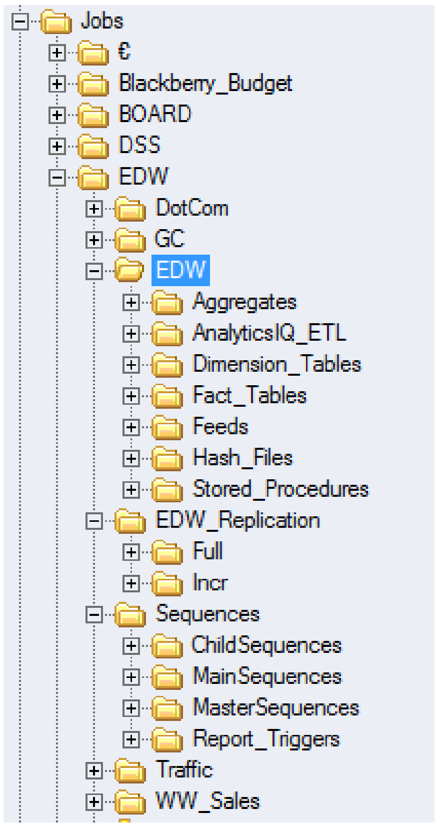
1. Interactively in the Designer, either in whole or via selection of individual objects.
2. Command Line on the client.
3. Deployment via the **Information Server Manager**. This approach also has some interesting documentation features.
4. On the server via the DSXImport Service.

Importing and exporting are very simple and mostly mechanical, so I will not go into the actual process, which is detailed in the KM20x course and product documentation. However, with project organization there are two main points to consider:

1. What defines a unit of work? Meaning how are objects saved so that when it is time to make the export or the package (depending on what tool you choose to use), how do you ensure that you get everything you need, but nothing unwanted? Treat the change management window as if someone else you don't know will be performing the movement of your code to the new environment, and that person is not very familiar with the included pieces. This saves you the effort of complex documentation that may or may not be understood correctly.
2. Environment configuration: In most cases there will be file paths, data sources, user names, passwords, etc. Ideally, and it's fairly easy today with the use of user-defined environment variables, objects should be able to drop into one environment from another and function with no modifications or configuration related to the release, unless something other than the code is changing. These environment variables and their values can be imported and exported via a file, making them versionable.

Project Organization

A mature DataStage project will often have hundreds, if not thousands, of jobs and ancillary components. Naming standards often determine whether you sink or swim. At the very least, they increase efficiency when it comes to revisiting a component at a later time. Readability is essential for you and the team(s). A strategy based on your needs will have to be developed. Organization is done in folders and subfolders, typically by unit of work. These units of work are often broken down into subcomponents, for example: jobs that refresh staging tables, jobs that load fact tables, dimension tables, call stored procedures, etc. For the most part, this is fairly straightforward to predict. The problem arises when there are shared objects that are used by multiple units of work. This complicates things when you want to change the shared object. One solution is to have different DataStage projects for individual units of work, essentially making things more modular—a code branch if you will. Another benefit is that different runtime users can be assigned to different projects, making the debugging and tuning, as well as identifying a process, easier. The following is an example of a series of folders that contain some 1,500 DataStage jobs. There are plenty of shared components. For example, let's say you have a job that invokes a report. This job will be parameterized and the parameter values will be passed at the point of invocation. If you decide to change this job in a future release, you will have to identify every place this job is used, and update how and what parameter values are passed. Shared objects are great, but they create additional dependencies.

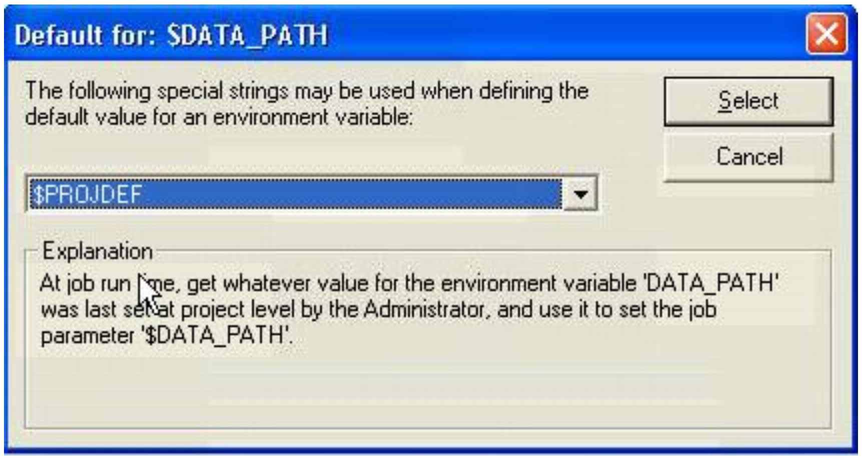


Environment Configuration Using Environment Variables and the DSENV

DataStage has always had the ability to have variables called parameters to pass values such as file paths, the value for a where clause, data source names, dates, etc., to a series of linked or subordinate jobs. This adds the necessary flexibility and portability we need at a minimum. Parameters are part of the job itself and don't always lend themselves to consistency or mass modifications. In the first IBM release, version 8.0, parameter sets were introduced. Although not the answer to everything, parameter sets are shareable objects, and certainly helped things. If you added a parameter to a parameter set in a subsequent release of your component, the new parameter would be available to all the jobs using that parameter set, but the job would not break if you didn't use the new parameter. This greatly increases consistency across all of your jobs.

Taking this to the next level, the parameter set can be passed values from DataStage environment variables. I don't mean to allude that these are the only options, but from a DataStage perspective, these are the more common approaches. It is certainly possible to use operating system-level environment variables. I have done this successfully in the past for directories, but it involves the administrator of the server. At the time, the mechanisms for handling the variables I will discuss had not yet been developed. For this conversation, let's limit our domain to things within the control of the DataStage admin. DataStage environment variables come in two forms: engine level and project level. Engine-level variables are stored in the dsenv file and affect all projects. Project-level variables are managed in the DataStage Administrator client. Although they are less global, they do come with the utility to be easily exported or imported along with their values. Another point to note is if a job(s) uses environment variables, those variables will typically be imported into the new environment as part of the job. The variable value will, however, be left blank. This ensures that the job will not compile or run, as it will not have a default value. This is better than running with the wrong value, so it's a good safeguard. All environment variables are prefaced with a $ sign. They can be overwritten at a lower level. For example, if you don't specify the variable, the job uses the default. $APT_CONFIG_FILE controls the level of parallelism and is in every project. If you don't specify it, you get the default value. If you do specify it, you can change its value at runtime. Such might be the case if you wanted to run a specific job with a different level of parallelism on certain days.

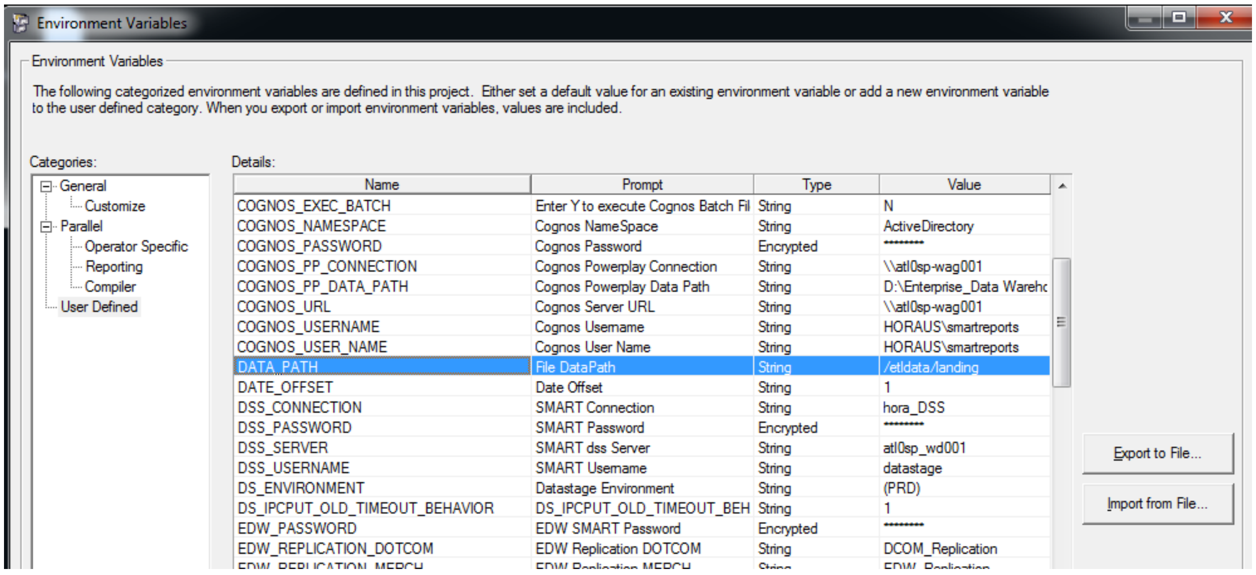
One subset of DataStage environment variables are user-defined environment variables, which can be very efficient and powerful. They function just like one of the predefined variables, but you get to choose the name and the value. These are used in the same way to add flexibility and portability to your components. Data sources, user names, passwords (encrypted), and file paths can all become part of the project on whatever server they reside. So, when you move a job(s) from one environment to another, the job will run with the values specified for that environment. This method is efficient and after the initial configuration, has fewer moving parts, as no subsequent management of these variables and their values are required unless there is a change to that environment. For example, suppose you had a variable for called $DATA_PATH that was used by hundreds of jobs and pointed to a certain file location. Over time, business is good, the volume of data has increased, and a new file server is implemented. Because the file location for all of these jobs is contained in this user-defined environment variable call $DATA_PATH, it can be changed in one place; and every place the $DATA_PATH variable is referenced, the new value will be used by the job, provided you have chosen to have the default value sourced from the project using the $PROJDEF specification.



The following special strings may be used when defining the default value for an environment variable:

At job run time, get whatever value for the environment variable 'DATA_PATH' was last set at project level by the Administrator, and use it to set the job parameter '$DATA_PATH'.

The following image shows the variable and its value from within the DataStage Administrator.



The following categorized environment variables are defined in this project. Either set a default value for an existing environment variable or add a new environment variable to the user defined category. When you export or import environment variables, values are included.

Name	Prompt	Type	Value
COGNOS_EXEC_BATCH	Enter Y to execute Cognos Batch Fil	String	N
COGNOS_NAMESPACE	Cognos NameSpace	String	ActiveDirectory
COGNOS_PASSWORD	Cognos Password	Encrypted	********
COGNOS_PP_CONNECTION	Cognos Powerplay Connection	String	\\atl0sp-wag001
COGNOS_PP_DATA_PATH	Cognos Powerplay Data Path	String	D:\Enterprise_Data Warehc
COGNOS_URL	Cognos Server URL	String	\\atl0sp-wag001
COGNOS_USERNAME	Cognos Username	String	HORAUS\smartreports
COGNOS_USER_NAME	Cognos User Name	String	HORAUS\smartreports
DATA_PATH	File DataPath	String	/etldata/landing
DATE_OFFSET	Date Offset	String	1
DSS_CONNECTION	SMART Connection	String	hora_DSS
DSS_PASSWORD	SMART Password	Encrypted	********
DSS_SERVER	SMART dss Server	String	atl0sp_wd001
DSS_USERNAME	SMART Username	String	datastage
DS_ENVIRONMENT	Datastage Environment	String	(PRD)
DS_IPCPUT_OLD_TIMEOUT_BEHAVIOR	DS_IPCPUT_OLD_TIMEOUT_BEH	String	1
EDW_PASSWORD	EDW SMART Password	Encrypted	********
EDW_REPLICATION_DOTCOM	EDW Replication DOTCOM	String	DCOM_Replication

Conclusion

I hope that you have gotten the impression that the DataStage environment is feature rich and, although probably not exactly what we might build today if we were to redesign the product from scratch, it is quite extensible and straightforward to those familiar with it. Through evolution the toolset has risen to many an occasion and has become prolific in several organizations. As is the case in any organization, communication and clear expectations will determine the ease and comfort of which success will be realized. My views were developed from my own experiences over some 15 years of DataStage development. But, there are many other views. I recommend the publicly available Redbook on DataStage Standard Practices (http://www.redbooks.ibm.com/abstracts/sg247830.html?Open) for additional content and recommendations from those that have walked the walk. Most importantly, you should develop a plan upfront. Try not to back yourself into a corner. No plan is perfect, and the ability to change based on assumptions that were not accurate or changing requirements is essential. If you get something to work and it is not an elegant solution, think about how it can be made one. Keep focused on portability and modularity. It is often much better in the long-term to have a series of smaller jobs, than one large job to accomplish the same tasks. While true that there may be a slight start-up penalty using this approach, maintenance, debugging, and often migration, will be simplified. Granularity leads to flexibility. Give thought to and develop naming standards. The aforementioned Redbook has many examples from experience. How you will version and organize your components and communicate your code drops to the other interested parties must be clearly understood by all interested parties. Most of the resistance I have experienced from other groups within an organization stem from a lack of understanding of DataStage. These groups are typically trying to ensure success and won't want to take a risk with something they don't understand fully. DataStage enjoys a large install base today, and it is growing under the broad reach of IBM solutions. But, it is still rather unknown by comparison. Prepare to present your approach to your stakeholders and get buy-in. People are often resistant to things they don't understand, so educate your stakeholders and let them form their own opinions. DataStage is a solid product deserving respect. Set the stage to make this happen. Show that you have done your homework and earn the title of a solution provider. This will make your relationship with DataStage and your organization a positive and fruitful one.

Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge through training.

IBM InfoSphere Information Server Administration V9.1 (KM502G)

IBM InfoSphere DataStage Essentials V11.3 (KM203G)

Visit **www.globalknowledge.com** or call **1-800-COURSES (1-800-268-7737)** to speak with a Global Knowledge training advisor.

About the Author

Greg Arkus joined Ardent Software via the acquisition of the Quality Manager product of which he was a principal engineer. The Quality Manager product has evolved into what is now Information Analyzer, which is a major application within the Information Server. From there, he became part of the Advanced Consulting Group, developing and applying many DataStage solutions ranging from web analytics to SAP and Siebel integration. Greg is certified in DataStage and Information Analyzer, has contributed test questions to several of the IBM certification exams, and co-authored a Redbook on Metadata Management using the Information Server. Prior to becoming a Global Knowledge instructor, he managed the training requirements for the worldwide Information Server practice at IBM. In his spare time, Greg likes to fly and races sailboats in the San Francisco bay area. He possesses airline transport pilot, flight instructor, and U.S. Coast Guard's Captain's licenses.

Copyright ©2015 Global Knowledge Training LLC. All rights reserved.