DMExpress and DMX-h

Implementation Best Practices

© Syncsort® Incorporated, 2015

All rights reserved. This document contains proprietary and confidential material, and is only for use by licensees of DMExpress. This publication may not be reproduced in whole or in part, in any form, except with written permission from Syncsort Incorporated. Syncsort is a registered trademark and DMExpress is a trademark of Syncsort, Incorporated. All other company and product names used herein may be the trademarks of their respective owners.

The accompanying DMExpress program and the related media, documentation, and materials ("Software") are protected by copyright law and international treaties. Unauthorized reproduction or distribution of the Software, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

The Software is a proprietary product of Syncsort Incorporated, but incorporates certain third-party components that are each subject to separate licenses and notice requirements. Note, however, that while these separate licenses cover the respective third-party components, they do not modify or form any part of Syncsort's SLA. Refer to the "Third-party license agreements" topic in the online help for copies of respective third-party license agreements referenced herein.

Table of Contents

1 Introduction ... 1

1.1 In this Document ... 1

2 DMExpress Software Management ... 2

2.1 Requirements ... 2

2.2 Installations and Upgrades ... 2

2.3 Software Versions ... 2

3 Project File Management ... 3

3.1 Application Files ... 3

3.1.1 Shared Files ... 3

3.1.2 Project-specific Files ... 4

3.2 Data and Log Files ... 5

3.2.1 Source and Target Data Files ... 6

3.3 File Naming Conventions ... 7

3.3.1 Jobs ... 7

3.3.2 Tasks ... 7

3.4 Source Control ... 8

3.4.1 Setup ... 8

3.4.2 Usage ... 8

4 Design and Development ... 10

4.1 Task Components ... 10

4.1.1 Sources ... 10

4.1.2 Targets ... 11

4.1.3 Intermediate Files ... 11

4.1.4 Task Types ... 11

4.1.5 Workspace ... 12

4.2 Process Flexibility ... 12

4.2.1 Jobs and Tasks ... 12

4.2.2 Source and Target Files ... 12

4.2.3 Intermediate Files ... 13

4.2.4 Connections ... 13

4.2.5 Other Flexibility Considerations ... 14

4.3 Error Processing ... 14

4.3.1 Data Exception Handling ... 14

4.3.2 Custom Task Exception Handling ... 14

4.3.3 Recoverability ... 14

4.3.4 Exception Notification ... 14

4.4 Custom Processing ... 15

4.5 DMX-h Processing Location ... 15

4.5.1 Server Processing vs. Cluster Processing ... 15

4.5.2 Map Side vs. Reduce Side Processing ... 15

5 Performance Optimization ... 17

5.1 Reduce Data Early, Expand Data Later ... 17

5.2 Minimize Read/Write Bottlenecks ... 17

5.3 Plan for Parallelism ... 18

5.4 Optimize CPU Processing ... 18

5.5 Define Data Properties ... 18

5.6 Minimize Job Customization ... 19

5.7 Optimize Task Output as Input to a Subsequent Join Task ... 19

5.8 Determine Lookup vs. Join ... 19

5.9 Optimize Database Access ... 19

5.9.1 Database Reads ... 20

5.9.2 Database Writes ... 20

5.10 Locale Conversion ... 21

5.11 Optimize Hadoop Configuration in YARN ... 21

5.11.1 Data Size ... 21

5.11.2 Memory ... 22

5.11.3 CPU ... 22

6 Documenting Jobs and Tasks ... 23

6.1 Comments ... 23

6.2 Naming and Readability ... 23

6.2.1 Fields ... 23

6.2.2 Values and Conditions ... 24

7 Testing, Debugging, and Logging ... 25

7.1 Testing ... 25

7.2 Debugging ... 25

7.3 Logging ... 26

7.3.1 DMExpress Server Logs ... 26

7.3.2 DMX-h Hadoop Logs ... 26

7.3.3 Log Maintenance ... 26

1 Introduction

DMExpress is Syncsort's high-performance data transformation product. With DMExpress you can design, schedule, and control all your data transformation applications from a simple graphical interface on your Windows desktop, as well as run them from the command line in a Unix/Linux environment.

DMX-h is the Hadoop-enabled edition of DMExpress that allows you to develop ETL applications in the DMExpress GUI or DTL language and run them seamlessly in the Hadoop MapReduce framework.

When designing, developing, testing, debugging, running, and maintaining DMExpress (and DMX-h) ETL solutions, there are many considerations for improving design-time and run-time efficiency, clarity, reusability, accuracy, and performance. Following the recommended best practice guidelines will improve your experience and outcome with DMExpress.

In this document, DMExpress will be used to refer to both DMExpress and DMX-h except where DMX-h-specific considerations are addressed.

1.1 In this Document

DMExpress best practice guidelines are provided for the following:

- DMExpress Software Management
- Project File Management
- Design
- Performance Optimization
- Documenting Jobs and Tasks
- Testing, Debugging, and Logging

2 DMExpress Software Management

DMExpress software consists of client and server components:

- The client component is an Integrated Development Environment (IDE) that is installed on Windows. The client IDE consists of a Job Editor and a Task Editor, which are used to develop and test the applications. The DMExpress Help is accessible only on the client.
- The server component can be installed on the same Windows machine as the client, on a different Windows machine, or on a UNIX/Linux machine. For Hadoop clusters, it needs to be installed on all of the nodes in the cluster. The server component is used to schedule and run the DMExpress applications.

2.1 Requirements

For information on the hardware, operating system, and software required for DMExpress, review the Platform, Package and Feature Support Matrix. To access the matrix, click on **Download** on the MySupport DMExpress Software Download page, or see the DMExpress help.

2.2 Installations and Upgrades

Read the DMExpress Installation Guide prior to planning your installations and upgrades. To access the guide, click on **Installation Instructions** on the MySupport DMExpress Software Download page, or see the DMExpress help.

2.3 Software Versions

The version of the DMExpress server software must be at least as high as the version of the DMExpress client software that is used to develop the jobs.

- As a best practice, install the same version of DMExpress on the client and on the server.
- If you cannot upgrade the server and the client at the same time, upgrade the server software prior to upgrading the client software.

3 Project File Management

Your projects will be easier to develop and maintain if you set up standard directory structures and naming conventions for application and data files.

3.1 Application Files

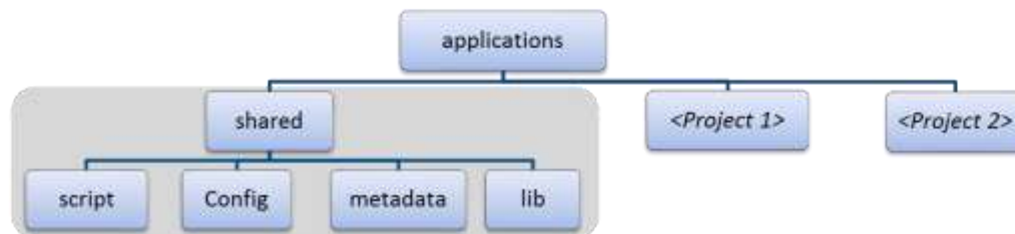
A DMExpress solution typically consists of a variety of files, such as job and task files, configuration files, scripts, Cobol copybooks, etc. Additionally, some of the files may be shared among several solutions.

It is highly recommended that you use a source control system for your application files, and it is imperative that you make frequent backups of either your source management projects or of your application files directly if not using source control.

3.1.1 Shared Files

Following are best practice guidelines for organization of shared application files:

- Create a central “applications” folder under which all the files across projects will be stored.
- Under the “applications” folder, create a “shared” folder to store the content which will be used by multiple projects. Consider categorizing the shared files by type as shown in the diagram and described in the table below.



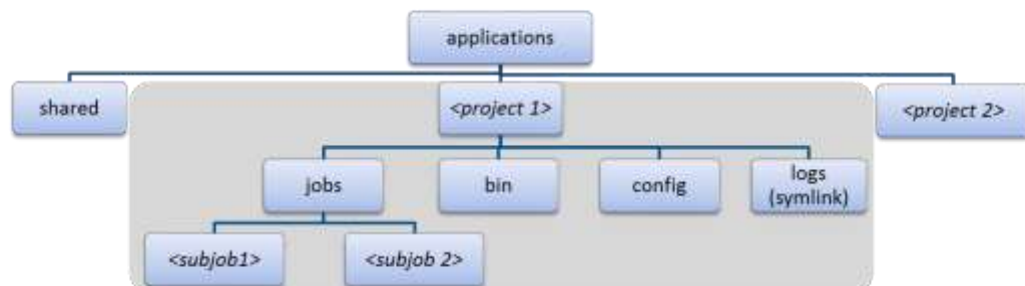
Description of Shared Sub-folders

Folder	Purpose
script	This folder will store scripts to be used across projects. For example: • an orchestration script which takes project name or code as argument, identifies the project location, sets global and project-specific variables, and triggers the main job of the project • the getLogs.sh script described in DMX-h Hadoop Logs. • a script to periodically archive/delete old log and data files according to your retention policy and disk usage tolerance
config	This folder will store global configuration files which need to be executed to set up the environment before any job runs in that system. For example: • configuration files for defining environment variables about the

Folder	Purpose
	Hadoop cluster such as namenode, edge node, etc. Consider having different versions for each type of environment, such as Dev, QA, UAT, and Prod.
metadata	This folder will store metadata shared across projects. For example: • Create a DMExpress task to store a list of database connections, remote file connections, HDFS connections, mainframe connections, etc. Any project which needs any connection information should link to this shared task to provide an easy way to manage connections. • Store layouts of third party file feeds used in multiple projects. • Identify common functions and expressions such as data validation rules, and store them as named values in a DMExpress task to be shared by different projects.
lib	Optional directory to store functions/libraries created in other programming languages. For example: • Custom functions that you want to call from DMExpress • Custom connectors to integrate DMExpress with other technologies

3.1.2 Project-specific Files

Files belonging to separate projects should be kept separate. We recommend creating project folders at the same level as “shared” under “applications” . A project will typically have many DMExpress job and task files, job-specific variables and parameters stored in configuration files, scripts, etc. To manage the storage of these files, consider creating the folders as shown in the diagram and described in the table below.



Description of Project Sub-folders

Folder	Purpose
Jobs	This folder will store DMExpress job and task files. For a small project with only a few jobs, you can keep them together in this folder. For larger projects, consider breaking this folder into subjob folders. Choose meaningful names when creating folders for job groupings:

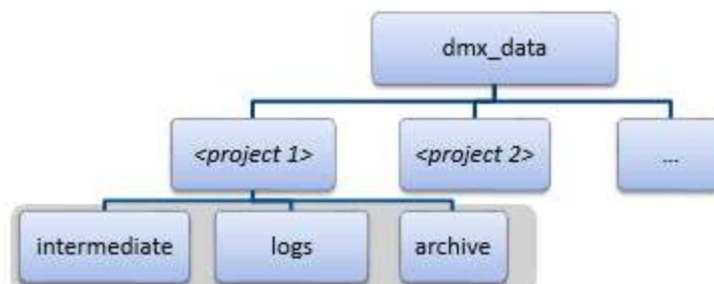
Folder	Purpose
	• Name according to the project or main line of business, for example “FinanceBilling”. • Subfolder names should be nouns representing the types of results expected, for example, “ReportingPreAggregates”.
Bin	This folder will store those project-specific executables There may be cases where a project uses batch scripts for certain purposes or a portion of a project can best be done in some other technology. For example: • Custom executables • Scripts to create folders/tables at runtime • Script to generate DTL dynamically • Scripts to orchestrate conditional sequencing or looping of jobs/tasks. Alternatively, this can be done using job customization in DMExpress.
config	This folder will store project-specific environment variables used by DMExpress tasks. Multiple configuration files can be created to run the same job with different parameters.
logs	Optional symlink to the folder where project logs are physically stored. This symlink will provide quick access to logs produced by the project.

3.2 Data and Log Files

Depending on job design, data and output files can include source, intermediate, workspace, target, and log files. Intermediate and target files may be overwritten or created as new. All of these files need to be stored properly and archived/deleted per the retention policy. Using a clear and consistent data directory structure and file naming convention ensures that:

- There are no filename conflicts between different jobs.
- There are no filename conflicts between multiple instances of the same job running concurrently.
- Logs can easily be located and identified when needed for analysis.
- Workspace contention is minimal. See section 4.1.5 for details on workspace.

To manage the storage of these files, consider creating the folders as shown in the diagram and described in the table below:



Description of Project Data Files

<table>
<tr><th>Folder</th><th>Purpose</th></tr>
<tr><td>intermediate</td><td>This folder will store all intermediate files produced by the project.

Intermediate files:
get created as DMExpress passes data between tasks, unless direct data flows are enabled.
are useful for job restartability, and may be unavoidable due to job design.
can be on the same device as sources and targets, but to avoid contention with workspace, should be written to a device different from the workspace.
are persisted data sets saved to the location specified in the task. These data sets may not be required after job execution, but are not automatically removed by DMExpress.
are temporarily stored in a folder created relative to the working directory on the mapper/reducer node for DMX-h MapReduce jobs. These folders are managed by the Hadoop framework, which removes them after the map/reduce process is complete.</td></tr>
<tr><td>logs</td><td>This folder will store the project logs. Server jobs and Hadoop jobs are different in terms of how many log files they produce and where they are stored. Server job logs can be written directly to this location, but a script is needed to pull Hadoop job logs here, as described in DMX-h Hadoop Logs.</td></tr>
<tr><td>archive</td><td>This is an optional folder for projects that have archival requirements such as the following:
Regulatory/compliance projects may require storing years' worth of final/target data.
You may want to store a few weeks/months of source files coming from a vendor/third party for tracing issues.
Keeping detailed historical data in tables is expensive. You may want to keep only current data in tables and create compressed files for historical data.</td></tr>
</table>

3.2.1 Source and Target Data Files

Sources and targets of a project can vary widely – they can be local or remote file server, database, mainframe, data from an external application, etc. Therefore, we recommend not to create source and target folders under each project unless you determine that folders will serve your needs. Following are best practice guidelines for managing source/target data files:

- Source and target data files can be anywhere, but to minimize contention, they should be on a device that is separate from the device specified for workspace.
- Review the speed of file read/writes on your devices as some file system choices, for example, networked file systems, are not optimal for performance. Sustained

sequential read/write throughput rates of 200 MB/sec or more are preferred. You can obtain an I/O benchmark test tool from Syncsort to verify read/write speeds.

3.3 File Naming Conventions

Following are best practice guidelines for standard naming conventions:

- Choose readable, meaningful names.
- Avoid punctuation such as dashes and other non-alphabetic characters.
- Use abbreviations sparingly and document accepted abbreviations.
- Use title case naming, in which the first letter in each word is capitalized regardless of significance.

3.3.1 Jobs

A project typically consists of a main job that triggers tasks and subjobs. All these jobs, whether created in the Job Editor or developed in DTL, are files stored in the *<project>*/jobs folder. Intuitive naming of DMExpress jobs should help differentiate them from other files and clearly identify their purpose, type, etc.

Our best practice recommendation is to create job file names in the following form, with the job parts described in the table below:

```
<job_type>_[C_]<job_name>.extension
```

Name Part	Description
<job_type>	`J`: Server or IEL Hadoop job `JMR`: User-defined MapReduce job `JM`: User-defined map only job
`C`	Indicates that the job includes job customization
<job_name>	Descriptive name that identifies the project or main line of business, such as “FinanceBilling”. Subfolders should be nouns named after the type of results expected, such as “ReportingPreAggregates”.
extension	`dxj`: Automatically given by the Job Editor `jdtl`: Identify the job as a DTL job

3.3.2 Tasks

The ETL logic of a job is built through a series of tasks. A properly named task should tell what that task does, how it was created, and where it executes.

Our best practice recommendation is to create task file names in the following form, with the name parts described in the table below:

```
<task_type>_<task_name>.extension
```

Name Part	Description
<task_type>	`T`: Server or IEL Hadoop task `TM`: User-defined map task `TR`: User-defined reduce task
<task_name>	Descriptive name that identifies the operation
extension	`dxt`: Automatically given by the Task Editor `tdtl`: Identify the task as a DTL task

3.4 Source Control

It is recommended to use a source control system for version management of your projects. DMExpress works well with most file-based version control systems, but an access-controlled system such as Subversion works best. Consider both setup and usage when choosing a source control solution. It also highly recommended that you create frequent backups of the source management project containing the solution files. See Integrating DMExpress with Version Control for more information.

3.4.1 Setup

Following are best practice guidelines for source control setup:

- Create projects for the jobs and tasks for different projects or lines of business in the organization.
- Create special projects for jobs and tasks that are reused across lines of businesses. In many systems, these projects can be included in other projects; for example, via external definitions in Subversion.
- Many source control systems allow customizable comparison or diff utilities. Consider using the dmxdiff utility to compare .dxj and .dxt files for version control clients installed on Windows.
- Ensure that all .dxj and .dxt files are configured as binary.
- Avoid using operations in the source control system that may cause corruption. Insertion of characters or modification of jobs and tasks may corrupt files. For example, .dxt and .dxj files cannot be merged automatically.

3.4.2 Usage

Following are best practice guidelines for source control usage:

- All developers create their own working directories for each of the projects on which they are working.
- The source management system tracks different revisions and allows the resynchronization of files.
- Locking the files that you are working on is the best way to avoid conflicts. If locking the files is not possible, conflicts may be identified between your file version and the latest file version using dmxdiff. You can manually merge the difference prior to file check in.

- After a version of the project is completed and tested, checkpoint the project so everyone knows the latest stable release of the project.
- Avoid using operations in the source control system that may cause modification to the jobs by the source control operations. This might cause the files to get corrupted.

4 Design and Development

When developing DMExpress solutions, it is beneficial to consider design strategies with respect to various aspects of the project before proceeding.

4.1 Task Components

4.1.1 Sources

The number of sources used provides an idea of job complexity. More sources typically means that more joins and lookups are used and, most likely, that more transformations are required to integrate the data. Following are best practice guidelines for source design:

- Understand all of your sources, including external data such as lookup tables, dimensions, etc.
- Transformations that pull in large amounts of data should be placed as late in the flow as possible. The size of the sources is a factor in the performance of the job.
- If all fields are not required for targets and transformations, eliminate them early in the process. In each of the sources, think about the fields that are required for processing.
- At the beginning of the job, use DMExpress to copy frequently used sources from remote servers or databases to the server running the job. In subsequent tasks, use the local copy to avoid bottlenecks caused by multiple transfers over the network.
- If multiple processes read from the same source, create a DMExpress reader task that reads from the source once and copies it to as many target files as needed. Each of these targets can be converted to a direct data flow and read by separate subsequent processes.
- Use direct data flows where possible.
- Only when it significantly reduces data, use user-defined SQL to filter on the where clause before data is extracted from the database and when indexes can be used to reduce the data extracted.
- Consider the metadata that must be collected, reused, or created for each of the sources. For sources that do not have bundled metadata, such as files, ensure that layouts are available.
- Consider linking external metadata for record layouts and database connections that are shared among tasks. By doing so, you can store encrypted passwords, avoid user errors, and make any needed changes in one place only.
- When defining sources for a join application, choose the small side for the left side. While not a requirement, this best practice is useful when data sizes cannot be determined or have not been estimated.
- Use delimited fields when processing data from a database that may contain nulls. To ensure that padding is removed from null fields, select **Compress all fields** on reformats.

4.1.2 Targets

Following are best practice guidelines for target design:

- When functional requirements require partitioned datasets, consider partitioning the data sets sooner in the job and create more parallel streams of tasks. A single final job target may indicate that the bulk of the transformations are applied in the single stream of tasks, whereas multiple targets can provide opportunities for parallel streams of tasks for transformations.
- When creating less optimal target formats such as XML, ensure that you move the data into the format as late in the process as possible.
- When final targets of the job are also intermediate flows between tasks, consider replicating the target in the task. One target flows directly to the next task and the other target serves as the final job target.
- Reducing data size or eliminating data earlier in your job is important to reduce read/write bottlenecks.
- Ensure that you know the fields required for your target and how these fields are transformed through the process.

4.1.3 Intermediate Files

Following are best practice guidelines for intermediate file design:

- Use a consistent record format for intermediate steps during processing.
- In general, choose the format that results in the shortest record. For example, using binary integers instead of decimals not only reduces the data transferred to each step, it may optimize operations that use those fields. However, if using binary integers means padding large text fields, the extra read/writes would negate gains from the optimized number formats.

4.1.4 Task Types

Understanding the relationships between your sources and targets provides the framework for your design and dictates the transformations required.

Copy

Copies can be used for the following:

- Applying business rules and filters when these rules and filters cannot be built into another transformation.
- General transformations that do not require a specific ordering on the data.
- Extracting data from sources and loading data to databases when restartability is important.
- Ingesting data into HDFS with DMX-h.

Aggregate

Data that must be summarized or de-duplicated requires aggregations to reduce data. Sorts can also be used when no summarization functions are required. When data is significantly reduced, an aggregate is preferred to a sort.

Join

Sources that relate to one another through primary and foreign key relationships require joins or lookups.

Merge

Consider merges when multiples sources must be brought together in the same sorted order in which they are sorted individually.

Sort

Data that relies on ordered processing requires sorts to order the process.

4.1.5 Workspace

Joins, aggregates, sorts, and tasks that extract large object data may require workspace. To avoid contention, workspace files should be located on a locally attached, high-performing storage file system, which is different from the file system that contains your source and target data.

See KB article 135 for recommendations on managing workspace for jobs run outside of Hadoop. When running in Hadoop, workspace is set automatically and typically does not need to be modified. However, if you need to change it, you can do so by setting `dmx.sortwork.dirs` as described in the “Running DMX-h ETL Jobs” help topic.

4.2 Process Flexibility

Using relative paths where applicable along with environment variables for entities that change in different environments – development, QA, and production – eases the migration of jobs from server to server.

4.2.1 Jobs and Tasks

- Use relative path specifications when adding tasks to a job and when adding linked metadata to tasks.
- When creating jobs and tasks, save them first before adding any content to ensure that all components relying on relative paths know the base directory.

4.2.2 Source and Target Files

- Use project-specific environment variables for the main source and target directories so they can be changed easily. You may want to reserve the use of the DMXDataDirectory environment variable for your intermediate data files, which are source and target files that are only used to connect tasks.
- For MapReduce jobs, the following environment variables are recommended:

Variable	Purpose
HDFS_SOURCE_DIR	Mapper (or reducer) HDFS input, for example: DMXhTestDrive:/MRLessons/HDFSData/Source
HDFS_TARGET_DIR	Mapper (or reducer) HDFS output, for example:

Variable	Purpose
	DMXhTestDrive:/MRLessons/HDFSData/Target
LOCAL_SOURCE_DIR	Mapper (or reducer) non-HDFS input, for example: DMXhTestDrive:/MRLessons/Data/Source
LOCAL_TARGET_DIR	Mapper (or reducer) non-HDFS output, for example: DMXhTestDrive:/MRLessons/Data/Target

- HDFS sources/targets require a remote HDFS connection. Specify the server using an environment variable, then set the variable as follows:
  - For runs on the ETL server, set it to empty.
  - For runs in Hadoop, set it to the namenode.

4.2.3 Intermediate Files

For Server Jobs

- Use relative path conventions to avoid filename conflicts across projects.
- If there's a need to run multiple instances of the same job simultaneously, consider:
  - using an environment variable in the filename to identify the instance. For instance, if you are running revenue aggregation for NJ and NY at the same time, specify the filename as RevAgg_${state}
  - appending a timestamp to the filename, for example, filename_${timestamp}

For User-defined MapReduce Jobs

- Use an environment variable such as $MAPRED_TEMP_DATA_DIR to specify the storage location for intermediate files.
  - For development and testing on the edge node, set it to a specific location.
  - For production, set it to dot (.), i.e. the current working directory.
- Conflicting filenames aren't an issue, as Hadoop creates a new working directory for each map/reduce job, and intermediate files are stored in a location relative to these working directories.

4.2.4 Connections

- If the same connection is used across multiple tasks and jobs, consider storing the connection in a common metadata task and linking to it.
  - If the connection user is always the same, store the password in the task for increased security (passwords stored in the task are encrypted) and ease of update.
  - If the users are different, consider having a copy of the connection metadata task for each user and avoid embedding it in every task that uses the connection.
- If a common user is not used for all submitters, an environment variable, which may not be secure at runtime, can be used, or the metadata task can be copied into the current project and modified to include the appropriate user information.

- Use environment variables for the components that define the connection. For example, for database connections, use an environment variable for the database name.

4.2.5 Other Flexibility Considerations

- Use environment variables for named values/conditions which can change per execution or as jobs migrate.
- Use environment variables for schema names when defining database source/target tables.
- For directory separators, use a forward slash, /, not a backslash, \, so that the job can be run on either Windows or Unix.

4.3 Error Processing

When designing your solution, you need to consider how errors will be reported and handled so that you can find, analyze, and correct them.

4.3.1 Data Exception Handling

DMExpress reports issues with fields that do not match data types during data transformation. Following are best practice guidelines for data exception handling design:

- In the Task Settings dialog, you can configure error reporting to provide the record or information about the record in question.
- While writing to database targets, you can change settings to abort processing when exceptions occur.
- If these are insufficient for exception handling, consider building validation rules into your tasks using functions such as IsValidNumber and IsValidDate.

4.3.2 Custom Task Exception Handling

When using custom tasks, ensure return codes indicate errors. DMExpress can be configured to abort when an exception occurs in processing.

4.3.3 Recoverability

The required level of recoverability in the event of a partial job failure determines where you create execution units in your job by disabling direct data flows. Disabling direct data flows causes data to be written to disk, which degrades performance due to the extra disk read/writes and the reduction of overlapping processing as the two tasks can no longer run in parallel.

4.3.4 Exception Notification

When submitting or scheduling with DMExpress, error conditions or successes can be sent as email notifications with log attachments.

4.4 Custom Processing

Often you may want to leverage existing processing already developed in your DMExpress job instead of re-implementing the specialty processing. Following are best practice guidelines for specialty processing design:

- For specialty processing that is performed in the database, for example, selection of data through an indexed query, consider using user-defined SQL in your tasks.
- For specialty transformations on files or tables using custom code, a script or program can be called as a custom task. Consider using standard input and output to pipe data in memory between tasks.
- For customized transformations on fields, consider using a custom-coded library that can be called from DMExpress via the Custom Function Framework.
- When customized task sequencing or looping is required, use the custom task as opposed to job customization.

4.5 DMX-h Processing Location

When running DMX-h jobs, you have the option to run the processing either on the edge node or in the cluster. When designing user-defined MapReduce jobs, you have the additional consideration of how to split the processing between the map and reduce sides.

4.5.1 Server Processing vs. Cluster Processing

Choose the edge node server to perform light operations such as the following:

- Data extract and load. If possible, avoid landing data on the edge node disk; use the external system and HDFS directly.
- Additional transformation tasks can be done during this copy process, such as the following quality assurance, filtering, and lookup tasks:
    - Data quality checks and filtering that require testing on individual records only
    - Confirmation of record counts
    - Lookups to static sources including database tables

CPU-intensive operations should be executed as MapReduce jobs in the cluster. For example:

- All joins
- Most sorts and aggregations (very small processes could possibly run on the edge node)

4.5.2 Map Side vs. Reduce Side Processing

User-defined MapReduce jobs provide the flexibility to choose between a map only or MapReduce job. It also provides the developer the control over what processing is done on the map vs. the reduce side. For best performance, any processing that can be done correctly on the map side should be done there. For example:

- Filtering
- Lookups
- Pre-aggregation

- Map-side join (only if size of small side will never exceed ½ of HDFS block size)

Any processing requiring comparison of multiple records must be done on the reduce side. For example:

- Large-to-large join
- CDC
- Final aggregation
- Database load (loads that generate table locking could cause deadlocks or additional delays)

5 Performance Optimization

While DMExpress automatically optimizes execution depending on the job and the environment, there are many ways to improve performance in the job design. DMExpress will issue run-time performance messages when it finds unexploited opportunities for performance improvement.

5.1 Reduce Data Early, Expand Data Later

Following are optimization guidelines for reducing data early and expanding data later:

- The smaller your data remains throughout the process, the more efficient your transformations will be.
- Change the order of tasks so the joins and aggregations that reduce data are closer to the beginning.
- Apply filters from the start to eliminate records as soon as possible.
- Only extract necessary fields from sources.
- Certain pivots and joins increase data, so place these tasks as close to the end as possible.

5.2 Minimize Read/Write Bottlenecks

Following are optimization guidelines for minimizing read/write bottlenecks:

- Use flat files instead of database tables for intermediate data.
  - In situations where the intermediate data is going to be used for subsequent join or lookup processing with a significantly larger table, consider using a temporary table for the intermediate data. In that case, push down the join or lookup processing into the database using user-defined SQL text and ensure the join or lookup key is an indexed database column in the large table. Benchmark this recommendation in your environment.
- If two tasks in the same flow require the same set of data, pass the data in direct data flows rather than having each task read the data.
- If you have enough free CPU resources, or for tasks that are not CPU intensive, consider compressing source data, target data, and workspace to reduce I/O time and the footprint on the disk.
- Workspace compression is specified via an option in the Run dialog:
  - It is unchecked by default for server jobs, but you should enable it for jobs which are not CPU intensive.
  - It is enabled by default for Hadoop cluster jobs.
  - When enabled, select the dynamic option, which allows DMExpress to choose the best combination of compressed and uncompressed workspace.
- It is best not to compress data flowing between tasks using direct data flows.

5.3 Plan for Parallelism

For non-DMX-h jobs, specify a higher maximum limit for memory in the Performance Tuning dialog to allow for greater parallelism and better runtimes when performing the following:

- Sorts when source data fits in memory.
- Joins when one side fits in memory.
- Aggregates when the target fits in memory.

Although rare, your system's processor and memory capacity might become highly utilized while running high data volume jobs/tasks in parallel. Under these circumstances, do the following:

- Create a file instead of a direct data flow so subsequent tasks cannot start until a previous task is complete.
- Use sequence arrows to run tasks sequentially to reduce the parallelism in a job.
- In the Performance Tuning dialog, use a memory limit specification to reduce the memory a task takes.

5.4 Optimize CPU Processing

Following are optimization guidelines for avoiding extra CPU processing:

- Analyze whether transformations are necessary and avoid duplicate processing.
- Presort is not required prior to a join or aggregation. If data must be sorted for a different purpose, the join or aggregation can leverage it when the data is identified as sorted.
- In general, partitioning is not required to gain performance as DMExpress internally processes in parallel.
- Avoid using the sort task type, which uses workspace. If the order is not important, use the copy task type.
- Use merges instead of sorts when bringing together presorted data.
- Avoid frequent data type conversions and keep data in formats that are optimal for the type of transformation being performed. For example, binary, double floats, and fixed length records are best for numerous calculations.
- In the Performance Tuning dialog, select **Original order of equal-keyed records need not be maintained** when possible.
- When doing so won't significantly increase data size, consider using fixed-length records, fixed-position fields, and fixed reformats.
- Use named values and conditions to reduce the number of times the values and conditions are evaluated during execution.

5.5 Define Data Properties

Following are optimization guidelines for defining data properties:

- If data comes from a pipe, a direct data flow, a remote server, a table not recently analyzed, or user-defined SQL, specify the approximate data size of that data in the Performance Tuning dialog and provide the estimated record size.

- In the Performance Tuning dialog, the most optimal aggregation algorithm is selected when you provide an estimate of the aggregation's resulting number of records.

5.6 Minimize Job Customization

If you customize your job using a scripting language, performance optimizing features may be disabled. Scripting may also limit the portability of your job.

Following are optimization guidelines for minimizing job customization:

- If DMExpress is unable to perform the processing, use custom tasks to invoke your custom processing.
- Isolate any customization and implement it in a sub-job, thereby eliminating the need to customize the main job. Alternatively, create sub-jobs for sets of tasks that do not require job customization so DMExpress can optimize for those sets.

5.7 Optimize Task Output as Input to a Subsequent Join Task

The output of a given task may become the input to a join task. When defining the column positioning order in the reformat of the given task's target, consider the following guidelines:

- To optimize the compare process, place the keys for the join first.
- Place non-key columns in the order in which they would be required for the output of the join.

5.8 Determine Lookup vs. Join

A look-up operation can be implemented in DMExpress through the lookup function or through a join task with one of the sources being the lookup source. When the target requires the combination of all matches of the same key on each side, the lookup function is not an option because the lookup returns only one match for the key. While the lookup function is simpler, using a join may improve performance in the following cases:

- Multiple lookup functions are defined to perform lookups from the same source using the same key field.
- If the size of your lookup source is a significant percentage of the physical memory on the machine. However, if the number of records in your task source is much smaller than in the lookup source, and the lookup source is a database table indexed on the lookup key fields, then a lookup function may be faster than a join.
- If you have a large number of lookups in a single task, consider distributing the lookup functions across multiple tasks to increase parallelism.

5.9 Optimize Database Access

When connecting to a database from DMExpress, ensure that you use the native DBMS selection for your database if DMExpress supports it. Avoid using ODBC or database utilities unless otherwise suggested in Syncsort knowledge base articles, product documentation, or by Syncsort Product Technical Support.

Using native connections provides the following benefits:

- speeds development
- consolidates tasks and logging
- avoids staging data on disk or in the database
- provides higher-performance extraction/load methods
- parallelizes extraction/load with transformation processing

5.9.1 Database Reads

Following are optimization guidelines for database reads:

- Ensure that the source databases tables are not heavily fragmented.
    - Reorganize the table and define a smaller number of LARGE extents.
    - Ensure that the source database has a sufficient amount of buffer pool and buffer cache.
- Provide information about estimated data sizes where the source tables are not recently analyzed. DMExpress uses the most optimal algorithms when it has information on the estimated data sizes of the sources.
- For Oracle databases, use the DMExpress fastest data extraction method.

5.9.2 Database Writes

Database writes are slower than database reads due to several factors such as writing to the buffer pool and data files; writing to the database undo space; writing to transaction logs; updating indexes; checking for constraints; firing triggers; and allocating free space.

Following are optimization guidelines for database writes:

- Create large tables with large INITIAL and NEXT extent sizes.
- If you can estimate the final size of the target table, pre-allocate the space to the table with a large initial extent before loading the data.
- If you are loading a significantly large amount of data into a large table and you have many indexes on the table, consider dropping the indexes before the load and recreating them after the load. For Oracle databases, consider using the NOLOGGING option during index recreation, which increases the rate of index creation.
- Consider using alternatives to database triggers on the target tables. For example, if a trigger is used to generate a natural key for a sequence/identity column, consider replacing the trigger with a DMExpress reformat using the TargetRecordNumber() function.
- Avoid using shorter commit intervals unless it creates transaction logging issues in the target database.
- For Oracle databases, use the DMExpress fastest load method.

5.10 Locale Conversion

Locale conversions may use excessive CPU and disable the ability to use high-performance joins. Following are best practice guidelines for locale conversion design:

- Know your source and target database system locales before designing your DMExpress jobs.
- Ensure that the data in the DMExpress transformations is compatible with the database locale. If the incoming data is:
  - ASCII, change the format of all the extracted text columns to “Treat as ASCII” at the time of extraction.
  - non-ASCII, but the majority of the data is:
    - 7-bit ASCII and many of the text fields are used with comparisons (filtering, sort fields, join fields, aggregate fields, and validation), consider extracting text columns as UTF-8 rather than locale.
    - not 7-bit ASCII, consider the size of the fields for the encoding in which you extract.
      - UTF-8 is an optimal comparison encoding, but may lead to much more data, which creates overhead.
      - Locale creates less optimal comparisons since system calls are used for comparisons, but can keep the data in a smaller set of bytes.
      - UTF-16 is a relatively efficient comparison encoding and the character is always 2 bytes.

5.11 Optimize Hadoop Configuration in YARN

DMX-h uses MapReduce to distribute processing. This MapReduce processing is managed by the YARN resource manager. Since the behavior and efficiency of DMX-h in the map and reduce phases are different than other MapReduce processes, we recommend larger resource settings to allow for faster and more efficient processing of larger quantities of data.

The map phase will typically process smaller sets of the data than the reduce phase. The reduce phase will typically perform more parallel processing than a mapper due to the relational processing more naturally performed in the reducer. This leads to more resource requirements in the reduce phases.

Hadoop properties used to control the allocation of resources to MapReduce jobs are typically set in the standard Hadoop configuration files for your distribution. With the exception of the block size property, which is a cluster-wide setting only, we recommend setting the properties mentioned below in the DMX-h job configuration file. Search “DMX_HADOOP_CONF_FILE” in the DMExpress Help for details on defining configuration properties per DMX-h job.

5.11.1 Data Size

Hadoop vendors like Cloudera recommend setting the block size such that mappers run between 40 seconds and 1 minute. Because DMX-h can efficiently process large amounts of data in a short elapsed time, the ideal block size for DMX-h is higher than the default block size on the different Hadoop distributions.

We recommend a block size of 512 MB – 1 GB for optimal performance. Alternatively, you can set the split size to 512 MB – 1 GB to allow more blocks per mapper. The block and split sizes can be controlled via the following Hadoop properties, respectively:

- `dfs.block.size` (this is a cluster-wide setting)
- `mapreduce.input.fileinputformat.split.minsize`

5.11.2 Memory

DMX-h memory usage will remain within the bounds of the JVM heap size. We recommend setting the mapper JVM heap size to 4 x split size or more, and the reducer JVM heap size to 8 x split size or more. For example, with a split size of 512 MB, set the map heap size to 2048 MB and the reduce heap size to 4096 MB.

The map and reduce JVM heap sizes can be controlled via the following Hadoop properties, respectively:

- `mapreduce.map.java.opts.max.heap`
- `mapreduce.reduce.java.opts.max.heap`

When setting the JVM heap size, it must not exceed the container limit. Consult your Hadoop distribution documentation for ratio recommendations. For example, Cloudera recommends keeping the JVM heap size at 80% of the container size. Following the above JVM heap sizes, that would mean a map container size of 2560 MB, and a reduce container size of 5120 MB.

The map and reduce container sizes can be controlled via the following Hadoop properties:

- `mapreduce.map.memory.mb`
- `mapreduce.reduce.memory.mb`

5.11.3 CPU

The number of vcores on the cluster corresponds to the CPU processing capacity of the cluster. Your Hadoop vendor will have specific recommendations on how to set the number of vcores relative to the number of cores and other resource factors.

The DMExpress engine utilizes multiple threads to parallelize its processing within a container. Multiple vcores allows for this processing to be more efficient. Since the reducer typically processes much more data than the mapper, and the reducer processing is more relational in nature, there is more opportunity for parallelization on the reducer side and therefore a reduce container would benefit from more vcores than the mapper.

We recommend setting the map vcores to 2 and the reduce vcores to 4. The number of map and reduce vcores can be controlled via the following Hadoop properties:

- `mapreduce.map.cpu.vcores`
- `mapreduce.reduce.cpu.vcores`

6 Documenting Jobs and Tasks

Jobs and tasks are often reused, modified, and debugged. Documentation is essential for future maintainability of jobs and tasks. Make your code easy to understand and "self-documenting" for the following reasons:

- Source data formats or business requirements can change.
- Unexpected output is generated or an unexpected error occurs.
- Someone must quickly figure out what a task is doing to update or debug it.
- The original developer is not available for consultation.

6.1 Comments

You can record comments in the following locations:

- Comments area in the Job or Task Editor via the **File > Properties** menu item.
- Add comments box. In the Job Editor, select **Edit > Add Comment**.
- An external file: text, MS Word, HTML, etc.

Comments recorded in a job always display in the log. Comments in a task optionally display per the setting in the Task Settings dialog.

Recommended documentation for all jobs:

- Using the comments box, add a comment under each task and subjob describing its purpose and function.
- Descriptions of environment variables, which must be set before execution.
- Names and descriptions of initial source and final target files and database tables.
- Information on any database connectivity requirements.
- Names and descriptions of any custom tasks.

6.2 Naming and Readability

In addition to commenting, you can make your applications self-documenting by providing meaningful names wherever objects are named. In addition to jobs and tasks as discussed previously, descriptive naming of sources/targets, fields/values/conditions, layouts/reformats, and connections will make your application much easier to understand and debug. Specific recommendations for Fields, Values, and Conditions follow.

6.2.1 Fields

When naming fields, choose names that describe the contents of the field such as "SalesAmount".

Name layouts according to the state of the information held in the data. For example, for the initial layout, use "SalesFactDataLayout"; for the final layout in which all dimensions are joined, use "SalesFactWithAllDimensionsLayout".

For reformats, provide a name for the target field that describes the contents of the field, which may not be the same as the value name mapped to that field.

6.2.2 Values and Conditions

The logic of your application is largely defined through the use of values and conditions in reformats, keys, and filters. Avoid using inline expressions in your tasks, as it results in logic that's hard to follow and causes needless re-evaluation of expressions when used in multiple places during execution. Instead, create named values and conditions, and reference them by name wherever needed in the task to improve both readability and performance.

Following are best practice guidelines for naming values and conditions:

- When naming values, use names descriptive of the operations performed and of the entity resulting from the operation. For example:
  - Use "FeeCalculation" for a value that calculates a fee.
  - Use "URLLowerCase" for a value that calls the ToLower() function on a URL.
  - Avoid vague prefixes like "new", as in "NewSalesAmount".
- When naming conditions, use names that assert the condition being checked to help conceptualize the data that results from a true condition. For example:
  - "SalesAmountIsNull"
  - "CustomerCodeInActiveSet"
- Break down complex expressions into multiple named values.
- Use spaces after commas between parameters of functions.
- Use spaces before and after operators (+, -, *, /, ||, =, >, <,).
- Format expressions using newlines and indentation to improve readability.
- Use C-style (/* … */) comments to explain complex expressions.

7 Testing, Debugging, and Logging

7.1 Testing

Following are best practice guidelines for testing:

- Test as you develop tasks.
- Always check logs for performance messages, even for successful runs.
- Use bulk filtering to test subsets of data.
    - To limit processing to the specific set of records that is causing a problem in a transformation, use a combination of skipping records and retaining a subset of records.
- Leverage DMExpress to check results.
    - If needed, use a join to compare two result sets.
    - Create tasks that validate the data.

Since MapReduce jobs don't show intermediate data, testing such jobs is more complicated. Use the following best practice guidelines for testing user-defined MapReduce jobs:

- Test on the edge node before submitting to the cluster.
    - Test with more than one reducer to confirm that mapper output determines the partition IDs correctly.
    - Intermediate data files and log files will remain for analysis.
- Capture meaningful fields in targets.
    - Information needed may get lost as it goes through intermediate data.
    - Utilize the Reformat task to capture problematic fields between the mapper and reducer.
- Apply filters to create a smaller version of the HDFS file for testing.

7.2 Debugging

Following are best practice guidelines for debugging DMExpress applications:

- Disable direct data flow so intermediate files are created and can be used to narrow down the issue.
- In case of an error or exception, modify the task settings so record contents are written in the logs.
    - Consider adding a record number to your data using target record numbering so that it's easier to identify the displayed records.
- Use reformats to display intermediate calculations and user-defined values.
- Perform Lineage Analysis and Impact Analysis in the Job Editor to determine the source of problems created through a series of transformations.
- Test-run individual tasks.
- Use filtering to eliminate "good" data so you can focus on problem data.
- Use data sampling from the Job or Task Editor to review sample data. You can view the data in hexadecimal in the Sample and Record Layout dialogs to understand the byte codes that make up the data.

7.3 Logging

When running jobs, DMExpress (as well as Hadoop when running DMX-h jobs) produces log files that report run-time results, warnings, and errors. These logs can be helpful in finding and correcting run-time issues.

7.3.1 DMExpress Server Logs

- DMExpress produces a single log file for each main job; task and subjob logs go in the main job log file.
- Always timestamp the log files, and include the job name in the log file name.
  - When running from the GUI, use the variables available for log file names such as %DMXJobName%, %DMXJobStartTime%, etc.
  - When running from the command line, create similar variables.
- Logs can be written directly to the *<project>/*logs folder.
  - When running from the GUI, the Run dialog provides an option to specify where you want the logs to be copied.
  - When running from the command line, logs go to standard error by default, but can be redirected to the project folder.

7.3.2 DMX-h Hadoop Logs

- Every map/reduce process produces a log file containing DMX-h and Hadoop framework logs. DMX-h also creates one server log which can be named and stored as described above.
- Map/reduce logs file naming is controlled by the Hadoop framework, which puts the mapper/reducer number and attempt number in the filename.
- Storage on individual nodes is also controlled by the Hadoop framework. Having the logs distributed across the cluster makes it difficult to perform log analysis. We recommend downloading the getLogs.sh script from KB article 261 to gather the logs and copy them to the <project>/logs folder. The script creates a folder by Hadoop job Id and puts all the map/reduce logs in that folder.

7.3.3 Log Maintenance

Keeping job statistics over time is useful when redeveloping a frequently-used process or trying to determine why a job's performance has declined. XML job logs can be processed and gleaned for statistics as well as errors, and can be analyzed or loaded to a database for further analysis and trending. Log files can also be important for auditing.

The DMExpress Server dialog Jobs tab can accumulate job logs over time. Consider archiving the logs either by loading them to a database or saving a copy on the file system and deleting them from the Server dialog console. You can create a copy of the log in a specific archive directory via the **Copy job log to** option in the Run dialog. You may want to create a script to periodically clean out your own repository of the log files based on your auditing and space needs.

About Syncsort

Syncsort provides enterprise software that allows organizations to collect, integrate, sort, and distribute more data in less time, with fewer resources and lower costs. Thousands of customers in more than 85 countries, including 87 of the Fortune 100 companies, use our fast and secure software to optimize and offload data processing workloads. Powering over 50% of the world's mainframes, Syncsort software provides specialized solutions spanning "Big Iron to Big Data", including next gen analytical platforms such as Hadoop, cloud, and Splunk. For more than 40 years, customers have turned to Syncsort's software and expertise to dramatically improve performance of their data processing environments, while reducing hardware and labor costs. Experience Syncsort at www.syncsort.com.

Syncsort Inc. *50 Tice Boulevard, Suite 250, Woodcliff Lake, NJ 07677* *201.930.8200*