

Data Integrator

Best Practices Handbook

Implementing Standards for Your Integration Project Team

Pervasive Software, Inc.
12365-B Riata Trace Parkway
Austin, Texas 78727 USA

Telephone: 888.296.5969 or 512.231.6000

Fax: 512.231.6010

Email: info@pervasiveintegration.com

Web: <http://www.pervasiveintegration.com>

PERVASIVE®

See copyrights.txt in the product installation directory for information on third-party and open source software components.

Best Practices Handbook

October 2010

Contents

1	Using Recommended Life Cycle Standards	1-1
	<i>Implementing Standards for Your Integration Project Team</i>	
	Scoping Your Integration Project	1-2
	Preparing for the Initial Project	1-2
	Planning Integration Design	1-2
	Naming Conventions for Specification Files, Variables, and Objects	1-4
	Files	1-4
	Variable and Functions	1-4
	Macros	1-5
	Setting Up Workspaces and Repositories	1-6
	Defining Workspaces	1-6
	Specifying Repositories	1-9
	Moving from Development to Test and Production Environments	1-11
	Before You Begin	1-11
	Development Environment	1-11
	Test Environment	1-14
	Production Environment	1-15
	Using Macros During Deployment	1-17
	Associating Macro Files with Engine Installations	1-17
	Using Macros to Specify File Paths	1-18
	Adding Macros to RIFL Scripts and Event Actions	1-19
	Macro Scenarios	1-22
2	Improving Performance	2-1
	<i>Design Choices to Improve Performance</i>	
	Enhancing Run-Time Performance	2-2
	Hardware	2-2
	Software	2-3
	Network Issues	2-3
	Designing Faster Transformations and Processes	2-5
	Design Tips	2-5
	Choosing a SQL Update Method	2-6
	Factors to Consider	2-7
	Using Process Designer to Run Maps on Multiple Threads	2-8
	Single Threaded Process	2-8
	Multithreaded Process	2-8
	Multi-Instance Threading	2-9
	Managing SQL Sessions in Process Designer	2-11
	Designing for Performance	2-11
	Achieving Best Performance with Large Files or Tables	2-15

Accessing Files or Tables	2-15
Accessing Tables.	2-16
Comparing Engine Configurations.	2-18
Business Case Scenarios.	2-19
Running on the Command Line to Improve Performance	2-25
3 Error Handling	3-1
<i>Tips for Handling Errors in Transformations and Processes</i>	
Error Handling and Trapping.	3-2
Setting Map Execution Properties	3-2
Selecting Map Error Logging Properties.	3-3
Using OnError Events	3-3
Catching Run-Time Errors	3-5
Error Handling Tips for Processes	3-7
Global Process Level	3-7
Process Step Evaluation.	3-8
Managing Log Files	3-13
Automated Log File Monitoring.	3-13
Automated Log File Archival.	3-14
Using the Error Log as a Troubleshooting Tool	3-15
A Appendix	A-1
<i>Supplemental Information</i>	
Automating Data Integrator Engine	A-2
User Scenarios	A-3
Choosing the Best Lookup Method.	A-5
Inline or Search and Replace	A-5
Static Flat File	A-6
SQL Pass-Through	A-8
Dynamic SQL	A-9
Incore Table Lookup	A-11
Generating Test Data Using the Null Connector	A-13
Comparing Unix and Windows Deployments.	A-14
Limitations.	A-14
Design Tips	A-14
Employing Code Modules and User-Defined Functions for Reusability	A-15
Code Module Use Cases	A-15
Creating Code Modules.	A-16
User-Defined Functions and RIFL Scripts	A-17
Creating User-Defined Functions	A-17
Using Global Variables	A-18
Troubleshooting Tips for Process Designer	A-19

Best Practices for Data Integration

This documentation suggests best practices for creating integration solutions. The topics lead you through the following project phases.

- Initial planning
- Using naming conventions for project files
- Managing metadata
- Setting up development, testing, and production environments
- Using macros in deployment and throughout transformations and process designs
- Improving transformation and process performance
- Error handling, trapping, logging, and notification

The following topics are also covered:

- “Using Recommended Life Cycle Standards”
- “Improving Performance”
- “Error Handling”

Using Recommended Life Cycle Standards

Implementing Standards for Your Integration Project Team

Your integration project can use these practices to increase the quality and raise the chances of success for your own solutions. This section covers the following topics:

- “Scoping Your Integration Project”
- “Naming Conventions for Specification Files, Variables, and Objects”
- “Setting Up Workspaces and Repositories”
- “Moving from Development to Test and Production Environments”
- “Using Macros During Deployment”

Scoping Your Integration Project

Before you start working with the integration tools, it is worthwhile to do some initial planning and preparation. When scoping a new data integration project, use the following checklist to determine how to begin.

Preparing for the Initial Project

Completed	Item
	Integration Type. Is your project primarily a migration, extract, transform and load (ETL), application integration, business to business transactions (B2B), data profiling, or other?
	Data Objects. What is the number of transformations, processes, source files or tables, target files or tables that you plan to use? Will this number vary between transformations and processes, or will it always be the same?
	Direction. Is the direction of your data connection one-way or bidirectional?
	Volume. What is the volume of your data? How many records do you plan to process? Find a unit of measure to count.
	Frequency. Do you want to run your processes and transformations for a single impromptu purpose, in scheduled engine runs, or continuously in real time?

Planning Integration Design

Once you are ready to use Data Integrator to develop your solution, follow this checklist.

Completed	Item
	Connectivity options. What source or target connector or component do you plan to use?
	Server address, User IDs, Passwords. Gather information on server addresses, and get user IDs and passwords ready.
	Shared Data Objects. Do you plan to use any code modules, RIFL scripts, SQL queries, or statements that were used in a previously designed transformation or process?

Completed	Item
	Shared Transformations or Processes. Are there any existing transformations or processes that you can use in the project?
	Data. Make a list of any data files, tables, or entities that you plan to use. Will the data require special handling, such as encoding or Unicode?
	Results Management. Do you need to build notifications of results or events, such as e-mail notification, custom log files, or data archival? Gather e-mail information and confirm checkpoints that require special logging.
	Identify Platform and Software Needs. What operating system platforms do you plan to use? Is client software needed for connectivity? Do you need special expertise to set up and configure the software which would require a database administrator or vendor technical services?

What to Do Next

“Naming Conventions for Specification Files, Variables, and Objects”

Naming Conventions for Specification Files, Variables, and Objects

Naming conventions make files and code more understandable by providing descriptive information about the file or code functions. Your team members can do their tasks more efficiently if they immediately understand what type of file or code is being used.

Files

- Append prefixes to specification file names based upon their type. For example, use “p_” for process, “m_” for map, “s_” for schema, and “j_” for join.
- Use descriptive file names that immediately tell team members and users about the file. Do not use spaces in file or folder names:
 - m_SAP_Cust_to_SFDC_Acct.map.xml
 - p_Migrate_QC_to_SFDC.ip.xml
 - s_SAP_Cust_Schema.ss.xml
- When multiple specification files are required to perform a single task or object migration, use a suffix. For example, m_SAP_Cust_to_SFDC_Acct_1.map.xml and m_SAP_Cust_to_SFDC_Acct_2.map.xml or m_SAP_Cust_A.map.xml and m_SAP_Cust_B.map.xml.
- When multiple versions of a specification file must be maintained within a single solution build, use a version suffix such as “p_Migrate_QC_to_SFDC_v2.ip.xml”.
- When naming the mapping step inside the process, attempt to make the step name match the underlying file name excluding the “m_”. So if the file name is m_SAP_Cust_to_SFDC_Acct.map.xml, use SAP_Cust_to_SFDC_Acct. If you want to point to multiple versions of a transformation in the same process, use unique names for each transformation step.

Variable and Functions

- For variable and function names, use mixed case with the first letter lowercase. Use meaningful names that could be readily understood by a team member or even someone outside the team, such as technical support. For instance, use v_balance for variables, or f_lookup for functions.

- Avoid one-character variable names except for temporary variables. Common temporary variable names are i, j, k, m, and n for integers, and c, d, and e for characters.
- Variable names used within RIFL script should also have a scope designation prefix for easy identification by the casual observer whether they are process-level (“p”), global (“g”), or local (no prefix). Mixed case should be maintained, for example pSomeProcessVar, gSomeGlobalVar, or localVar.
- DJMessage variable names do not require scope designation but should in most cases match the actual message name and should have a “Msg” suffix, for example: Set requestMsg = new DJMessage(“requestMsg”).

Macros

- Use all capitals and underscores when naming constants, for example MY_CONSTANT. Use this practice for macro names, such as the following common macro names:
 - XMLDB, DATA, LOGS
 - SQL_SVR, SQL_DB, SQL_USR, SQL_PWD
 - ORA_SID, ORA_USR, ORA_PWD
 - SF_USR, SF_PWD



Note Use Java naming conventions as defined by Sun for all other development purposes.

What to Do Next

“Setting Up Workspaces and Repositories”

Setting Up Workspaces and Repositories

Repository Explorer allows you to organize your workspaces and repositories and launch the design tools. To organize transformation and process files, at least one workspace and repository must be defined.

When you first launch the Repository Explorer, it creates a set of default folders. The workspace root directory is created with the user's login name. It contains a Workspace1 subdirectory, which is the default workspace location, and an xmldb subdirectory for the default repository. The default directory structure is as follows.



Keep this default repository even if you want to add additional repositories. This is especially important if you want to add repositories on a network server. If the network server goes down, you can still do development or testing on your local default repository.

While this directory structure works for many users, you may want more control over your transformation and process files as you increase the number or scope of your projects. To do this, you can create additional workspaces or repositories.

You first need to determine if you need to use more than one workspace, or if you can use one workspace and multiple repositories. To make your decision, see the following sections.

- “Defining Workspaces”
- “Specifying Repositories”

Defining Workspaces

A workspace is a virtual location that allows you to organize work within the development environment. A workspace root directory is defined and contains one or more directories that can be set as workspaces.

Using the default example, the UserName directory is the root and the Workspace1 directory is the actual workspace. The workspace

folder contains your macro definition file, repository list file, and common files such as .rifl and .cxl files.

Your default installation includes only one workspace. This is because Repository Explorer references only one workspace at a time. Each workspace contains its own set of repositories and macro definition files. Many users do not need to create additional workspaces. For instance, if all of your projects reference the same macros, you can work with only one workspace. You can add new repositories for each new integration project. For more details, see “Specifying Repositories”.

Workspace Design Recommendations

It is recommended that each workspace be unique to a user and defined as a local directory. Using this practice helps you to avoid one user overwriting the repository list file of another user.

Follow these recommendations if you plan to create more than one workspace in your integration project. Each user on the team should do the following.

- Define their own local workspace.
- Create unique macros that reference the local workspace.
- Use those unique macro names when collaborating with other users.

For more information, see “Using Macros During Deployment”.

A good rule of thumb is to create an additional workspace if you need to separate macro definitions into different files, as in the following example.

Example Workspace “Phoenix”

In this example, all map and process specification files for a project called “Phoenix” share macro values and code modules. None of these files are used for other projects. So in this case, it is logical to define a separate workspace for project Phoenix.

Workspaces		Repositories
Workspace	Description	
Workspace1	default workspace	
Phoenix	Phoenix project workspace ...	

The workspace folder is a default location for specific files, such as user-defined connections, external code modules, or RIFL script files.

The Phoenix project example defines an entire folder structure for a team integration project. This project involves more than one phase and uses multiple repositories.

Phase One

The first phase is a CRM synchronization effort that requires several processes, transformations, schemas, RIFL scripts, and lookup files. Therefore, use the following directory structure to organize the project.

```
C:\Documents and Settings\UserName\Phoenix\CRM_Synch
```

In this example, the workspace root directory is UserName and the workspace folder is Phoenix. A workspace is added to use a separate macro definition file located in the Phoenix folder.

All files are stored in a repository directory named CRM_Synch in the Phoenix project folder. The CRM_Synch directory contains the following collection folders:

- Processes
- Lookups
- RIFLScripts
- Transformations

Create collections for each type to make it easier for developers to locate the files.

Phase Two

You can use Repository Explorer to switch back and forth between roles by redefining the root directory.

For example, phase two of the Phoenix project includes migrating data from one accounting system to another. You do not need access to all Phoenix specification files, only those associated with the accounting system data migration. To do this, use Repository Explorer to define a repository that points to the Accounting_Migration directory.

When you want to access all subdirectories again, redefine the repository in Repository Explorer to point to the higher level directory Phoenix.

Specifying Repositories

Depending on your integration project, you may need to group specification files into a single repository, or use multiple repositories. A repository can be any workstation or network directory available to the user. The following table details a local workspace folder named development and two network repositories named test and production.

Name	Root Location	Type
development	C:\Documents and Settings\username\dev	FILESYSTEM
test	//networkA/username/integration/test	FILESYSTEM
production	//networkB/username/integration/production	FILESYSTEM



Tip When you add or modify repositories, change the default name xmldb01 to something more descriptive, such as the name of your integration project.

In Repository Explorer, the display of several hundred specification files can cause performance issues. Therefore, it is good practice to add, remove, or modify specification files during development. If you are working on one project and need to switch to another, simply modify the repository and point to a different directory.

Repository Manager searches only one repository at a time, so you should define your repositories so that all relevant specification files are in the same folder. Use a root folder as the repository and then separate specification files in subfolders. For example, you can separate structured schemas into one subfolder and RIFL script files in another. This makes it easier to find the specification files you need during the design phase of a project.

Collections

A collection is a subfolder within a given repository folder. Collections store files relevant to your project that may or may not

be .xml specification files. For instance, a collection folder could be called “Data”, and include source and target files.

The Data Integrator engine usually accesses process designs first in a command line execution. To improve performance, store processes at the top level of the repository hierarchy and then include the subordinate specification files, such as transformation and schema files. This practice also allows you to more easily identify project repositories to package into .djar files in the Repository Manager interface.

Example of a Workspace, Repository, and Collection Files

The following hierarchy shows a process specification file at the top and subordinate files in collections within the same repository.

- **Project Workspace**
- **Project Repository**
 - Process specification file

Three Separate Collections

- Lookup files, RIFL scripts that contain user-defined functions
- Structured schema specification files
- Transformation specification files

See Also

Search for the keywords “Introducing Repositories, Workspaces, and Collections” in the online help.

Search for the keywords “Using Workspace Manager” in the online help.

What to Do Next

“Moving from Development to Test and Production Environments”

Moving from Development to Test and Production Environments

One of the most important tasks in an integration project is moving your specification files through the life cycle phases of development, testing, and production. If your team follows these guidelines, your progress should be smooth and efficient.

The following are recommended characteristics for development, testing, and production.

- All project team members understand and follow the process of testing, tracking changes, and migrating specification files from development and testing to production.
- During the development process, your team develops and modifies code without affecting existing production runs.
- The team migrates transformations and processes from development and testing to the production environment without altering or retesting the code.

Before You Begin

Important! A highly suggested prerequisite before you begin working through the life cycles is to set up macros. See “Using Macros During Deployment” and then return to this topic.

Development Environment

Installation—Install the Data Integrator design tools to create map and process specification files.

Before promoting specification files to the test environment, the developers on the project should conduct basic unit testing using the design tool user interfaces.

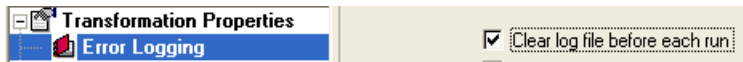
Development Life Cycle Tips

Use the following best practices for successful implementation of an integration project.

Practice	Benefit
<p>Do development work only in test environments that are separate from production environments. The only exception is if a production environment has not been rolled out yet.</p>	<p>You want to safeguard production data from any negative impact, and no need to stop production to make code changes.</p>
<p>Each distinct project should have its own workspace and a macrodef.xml file.</p>	<p>The macrodef.xml file stores paths and specific information, such as usernames and passwords, for a specific workspace. When you need to access different paths and information, you should create a new workspace and macrodef.xml file. See “Setting Up Workspaces and Repositories”.</p>
<p>Use comments whenever possible. For example, you can write comments in the following places:</p> <ul style="list-style-type: none"> • Macros • Map Properties • Source and target connection properties • Target Field Expressions • RIFL scripts 	<p>Comments help others understand your design.</p>
<p>Use a combination of whitespace and comment lines to break all code into smaller logical blocks. A logical block of code should not exceed 15 lines. Do the same for functions – if a function exceeds 15 lines of code, consider breaking it up into subfunctions.</p>	<p>Small blocks of code are easier to read.</p>
<p>Initialize all global map variables and objects in BeforeTransformation events and release them in AfterTransformation events.</p>	<p>Keep the variables in a standard location where they are easy to find. Release the memory allocated to the global variables to avoid memory leakage.</p>
<p>All process level variables and objects should be initialized in the Start Step and released in the Stop Step. Do not rely on the assumption that process level variables are automatically released and therefore not code for this in the Stop step.</p>	<p>Keep the variables and objects in a standard location where they are easy to find. Release the memory allocated to the process variables and objects to avoid memory leakage.</p>

Practice	Benefit
<p>Centralize RIFL scripting. In maps, calculate derived field level values, such as lookups and calculations, in an Execute step. Pass values to fields using variables rather than performing calculations within each individual field.</p>	<p>Execute scripts once and reuse the same value. Also, when someone else on your team wants to make changes, they can find and test the results faster than looking in individual fields.</p>
<p>Use DJImport objects and manual incore tables for lookups involving dynamic data. You can use the Lookup and TLookup functions, but remember that they can only be used with static data.</p>	<p>Lookup scripts port with the map, so there is no need to migrate lookup source files. Incore tables place data in memory and search from there, making searches faster and more efficient. However, if your RAM is insufficient, try another lookup method. For more details, see “Choosing the Best Lookup Method”.</p>
<p>When creating a new transformation, open Transformation and Map Properties > Execution Properties. Change the Source Schema Mismatch and Target Schema Mismatch default settings of Treat as Error to Use Connection, Map by Name.</p>	<p>When you alter source and target schemas in your transformations, you do not encounter mismatch errors.</p>
<p>Create and test at least 10 records for each possible use case during development phase. Run the tests with each code change.</p>	<p>Find errors and fix them in the development phase, so there are fewer errors encountered in the test phase. Also, running incremental tests after each code change makes spotting code that caused an error quicker and easier, saving significant development and testing time.</p>
<p>Store macro files, engine .ini files, .bat files, and other miscellaneous required files in \$(DJWORKSPACE).</p>	<p>Storing executable files in the root workspace results in improved run time performance.</p>
<p>Store project specification files in \$(DJWORKSPACE)\xml*. Follow the file naming conventions in “Naming Conventions for Specification Files, Variables, and Objects”.</p>	<p>Results in improved run time performance since process specification files are accessed before subordinate files, such as maps and schemas.</p>
<p>Store all project data files in \$(DJWORKSPACE)\data. Keep all temporary project data files used for batching in a subfolder called ‘temp’. Store test data in a subfolder called ‘test’. Use a DATA macro to switch between test and production data.</p>	<p>Using standard subfolders helps your team quickly find the files they need.</p>

Practice	Benefit
Store project log files in \$(D:\WORKSPACE)\log\ . Periodically copy the log files into a subfolder called archive.	You can easily find the log files when you need to refer to them. Keep an archive of older logs in case you need to send them to the system administrator.
In the Map Designer Transformation and Map Properties, select Error Logging . Then select Clear log file before each run .	Once you have run many transformations, the error log can include many messages. You cannot easily find messages for the last transformation that you ran. Clearing the log alleviates this issue.



Test Environment

Installation—Install the run-time Data Integrator Engine.

Once your team has created the specification files and performed unit tests in the development environment, you can move the specification files to a separate environment configured for testing and quality assurance.

In this environment, the primary source and target data connections are no longer development unit test instances, but are more rigorously controlled testing instances that mimic production data stores. Once you begin testing, try to simulate the production environment as closely as possible.

For details on creating test data, see “Generating Test Data Using the Null Connector”.

Using .Djar Files to Move Multiple Specification Files

A large, complex process can result in dozens of specification files from various design tools. To help ease deployment, you can package files into a .djar file in a Repository Manager utility.

A .djar file is a single file that contains a master process and all its dependents, including:

- Subprocesses
- Maps
- Transformations

- Schemas
- Code modules
- Lookup files

Processes and transformations can contain full path references to subordinate files such as lookup and code module files. These package files can be copied and run from anywhere.

After unit testing, create a .djar file to move specification files to the test environment and then to the production environment. Create different versions of these packages to maintain multiple instances of them based upon iterations of development.

Consider the package as a one-way process because once the package is created, you should not extract specification files from it and make modifications to them. Instead, modify the original specification files and create a new package.

For more information on creating .djar files using Repository Manager, search for the keywords “creating” and “package” in the online help.

See Also

“Generating Test Data Using the Null Connector”

Production Environment

Installation—Data Integrator engine is installed and requires connectivity to live data.

After testing and analysis, once specification files produce predictable results, deploy them to the production environment.

Production Life Cycle Tips

- Do not make any development or code changes in a production environment. If this is unavoidable, maintain a log of all differences between development and production environments. Maintain this log in a project report.
- Use a separate .ini file for engine execution that points to an engine license. Name the file dj800Eng.ini for 8.x products or cosmosEng.ini for 9.x products. Document any nondefault entries in the .ini file in a project report.

- Your deployment to production should include testing within the production environment. Create and test at least two records for each defined use case. Store test data in $\$(DATA)test\$. Otherwise, make these records clearly identifiable for easy removal later.

Using Macros During Deployment

Macros are symbolic names assigned to text strings, usually in file paths. You can use macros as a tool as you move integration project files from one life cycle environment to the next.

A macro definition file is an XML file that contains name-value pairs. These names can be used throughout a map or process to provide connection information. For example, a macro name can be substituted in the following connector options:

- Server
- Database
- User ID
- Password
- File or table connection paths

The following sections lead you through the process of using macros.

- “Associating Macro Files with Engine Installations”
- “Using Macros to Specify File Paths”
- “Adding Macros to RIFL Scripts and Event Actions”
- “Macro Scenarios”

Because developers of integration processes use macro names and system administrators use macro values, it is important to define a set of standards and conventions. Assign one person to keep the list of macro names and values for the project.

Associating Macro Files with Engine Installations

You can associate a macro file with an engine installation by using its .ini file. Once you type code in the engine .ini file, you do not have to type the -mf option on the command line each time you want to reference a macro.

➤ **To add a macro to an engine cosmos.ini file**

- 1 Open the cosmos.ini file in a text editor.
- 2 Add a variable in the [User Info] section.

```
[UserInfo]
macrofile=C:\directory\macrofilename.xml
```

- 3 Save and close the file.



Note You must repeat this process in each .ini file for the development, testing, and production environments.

Using Macros to Specify File Paths

The transformation in a process can remain unmodified and simply use the macro definition file in each life cycle environment because each environment has its own macro definition file. See the following examples.

Life Cycle	Macro Name	Macro Value	Remarks
Development	\$(ServerPath)	\\mydevspot\testdata	The original macro value in the development life cycle.
Test	\$(ServerPath)	\\testserver\testdata	Change the development macro value to the test macro value.
Production	\$(ServerPath)	\\productionserver\data	Change the test macro value to the production value productionserver.

Concatenating Macros

You can use two macros together.

```
$(server)$(directory)filename.txt
```

Or you can combine a file path and macro name.

```
\\prodserver\$(directory)filename.txt
```

Dynamically Defining Macros

Macros can be substituted for source and target connections and for properties that require a path, including:

- Log files
- Reject files
- Code modules
- Lookup files

Using the available macro functions, such as DefineMacro and MacroValue, you can minimize or eliminate the need to reference values in macro definition files. You can dynamically define macros in RIFL expressions using the DefineMacro function. In the

following example, a macro named TARGET contains the path name <drive>:\data\target\newfile.asc.

```
If MyCount = 1000 Then
DefineMacro("TARGET", "C:\data\target\newfile.asc")
End If
```

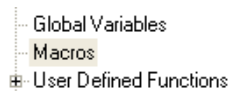
Once MyCount reaches 1000 during a transformation, TARGET is dynamically changed but records are still written to the original target file. The next time the transformation runs, the target file is named **newfile.asc**.

See Also

For more details, search for the following keywords in the online help: DefineMacro, ClearMacro, IsMacroDefined, MacroValue, and MacroExpand.

Adding Macros to RIFL Scripts and Event Actions

In RIFL Script Editor, the bottom left pane has a Macros node.



When you select this node, all macros in the current workspace appear in the bottom right pane. From that pane, you can select a macro to view its value and add it to a RIFL expression.

You can use macros in any RIFL script that requires a connection string. In fact, you can use macros for any name-value pair that you want to control external to the process. For the syntax, search for the keywords “Macro Manager” in the online help.

➤ **To add macros to ChangeSource and ChangeTarget actions**

- 1 In Map Designer, select **Tools > Define Macros > New**.
- 2 Enter the macro **Name**, **Value**, and **Description**.
- 3 Click **OK** to add the new macro and click **Close**.
- 4 Select **View > Transformation and Map Properties**.
- 5 Select an event handler.
- 6 Click the ellipsis in the **Action** column. The event window opens.

- 7 In the event window, click the **Action Name** arrow and select **ChangeSource** or **ChangeTarget**.
- 8 Click the **Connection String** ellipsis. The RIFL Script Editor window opens.
- 9 Enter the macro name you created in step 2 as the following:

```
+File=" & MacroExpand ("$(Temp) ")
```

or use this syntax:

```
MacroExpand ("+File=$(Temp) ")
```

Using Macros to Store Expressions

Macros can contain a single line of RIFL code, which is especially useful if this code is used repeatedly throughout many transformations. You can change the macro outside of Map Designer, such as on the Data Integrator engine command line or in Process Designer. Macros allow you to redefine transformation events with a single command.

Example

A macro named REJECTS is defined in Macro Manager as follows.

```
If Trim(Fields("Field3"))=="X" Then Reject
```

'Use the REJECTS macro in an AfterEveryRecord event.

```
Eval (MacroValue ("REJECTS"))
```

Every record is rejected that contains an X in the third field. For details on the individual functions, search for the keywords "Trim", "MacroValue", and "Eval" in the online help.

To run the transformation on the command line, use the option **Define_Macro NAME=value** to change the REJECTS value to the required value for that specific run.

'Reject every record with Cancel in the third field.

```
If Fields("Field3")== "Cancel" Then Reject
```

Testing Macro Return Values

To test macro return values, use the LogMessage function as in the following sample code.

```
LogMessage (MacroExpand ("$(myMacro) ") )
```

```
LogMessage (MacroValue ("myMacro"))
```

You can review the log file and track macro value changes throughout a transformation when the .rifl code changes periodically.

Escape Character for Macros

You want to log on to a web service with a user name and the password string "\$@ssw0rd". To prevent Map Designer from mistaking the string for a macro and trying unsuccessfully to expand it, add a second dollar sign as an escape character. Map Designer then interprets "\$\$(p@ssw0rd)" as "\$@ssw0rd" and sends the correct password string.

Using Incore Lookups

Build an Incore lookup using the Lookup Wizard. Edit the RIFL script, replacing MacroExpand("mymacro_name") with the following code.

Incore example using MacroExpand Function

```
myObjVar.ConnectionString =
MacroExpand("Server='$(SQL_Server)';Database='$(SQL_DB)
';UserId='$(SQL_UserID)';Password=$(SQL_PW) ")
```

Incore example using Dynamic SQL

```
myObjVar.ConnectionString =
MacroExpand("Server='$(SQL_Server)';Database='$(SQL_DB)
';UserId='$(SQL_UserID)';Password=$(SQL_PW) ")
```

Command Line Examples

In these examples, macros on the command line call macros in your transformations and processes.

➤ To use macros on the command line

Use the -mf option and include a path to the macrodef.xml file.

```
djengine -mf [path]\macrodef.xml $(Temp)\myMap.tf.xml
```

➤ To define a macro dynamically on the command line

Use the -D option to define any macro named "test" in a process.

```
djengine -pe -D test=mymacro MacroTestProcess.ip.xml
```

➤ **To use a variable on the command line**

First, define the variable myValue in **View > Transformation and Map Properties > Global Variables**.

```
set var=myValue
```

Then type the following code on the command line.

```
djengine -define_macro MyMacroName=%var%  
myTransformation.tf.xml
```

Macro Scenarios

Scenario 1 – Different Test and Production Environments

Set up one workspace as a test environment and another workspace as a production environment. Until you are sure your maps are running correctly, use files in the test environment to run the transformations. Your target file named mytarget.asc resides on both the test and production systems.

Location of test target file:

C:\testenvironment\targetfiles\target.asc

Location of production target file:

H:\production\target.asc

- 1** Define a macro called TARGETDIR with the value
C:\testenvironment\targetfiles.
- 2** Create a map.
- 3** On the Target Connection tab, instead of entering
C:\testenvironment\targetfiles\mytarget.asc as your
target file, you simply enter \$(TARGETDIR)\target.asc.

When you move to the production environment, you do not need to redesign your transformations. The production workspace has its own macro file, so you can set up a TARGETDIR macro there with the value H:\production, pointing to the production workspace.

All transformations in the production workspace that use the \$(TARGETDIR) macro point to the H:\production subdirectory instead of the C:\testenvironment\targetfiles directory defined in the test environment.

Scenario 2 – Two Computers, Different Directory Structures

You have designed transformations to be used within a process on your computer, but you decide to run the transformations on a team member's computer. The same source and target files are on each computer, but the directory structures are different.

Without Macros

You have to find the paths that will be used on your colleague's computer, open every transformation, change any path that does not match your colleague's paths, then resave and possibly rename the transformations. Alternatively, you can change the directory structure on one of the computers.

With Macros

Substitute macros for the file paths that point to the source and target files and give your colleague the names of the macros to set up before running the transformation.

See Also

For information on basic macro usage and syntax, search for the keywords "Macro Manager" in the online help.

For details on how to use macros with the DJMessage object, search for the keywords "DJMessage Object Type" in the online help.

Improving Performance

Design Choices to Improve Performance

This section covers the following topics:

- “Enhancing Run-Time Performance”
- “Designing Faster Transformations and Processes”
- “Using Process Designer to Run Maps on Multiple Threads”
- “Managing SQL Sessions in Process Designer”
- “Achieving Best Performance with Large Files or Tables”
- “Comparing Engine Configurations”
- “Running on the Command Line to Improve Performance”

Enhancing Run-Time Performance

Data integration run time may lengthen for several reasons:

- Large source files
- Use of many conditional expressions
- Conflicting settings in the transformation properties
- Many connections to and from a database where you must connect and disconnect to different tables

This section helps you reduce these performance problems.

To begin investigating, start with the basics. Review the possibilities of hardware, software, and network issues. Several fine-tuning suggestions are included.

Hardware

- **System Requirements.** First, make sure that your system meets the system requirements. Search for the keywords “minimum system requirements” in the online help.
- **Available physical memory and swap file usage.** If there is low available physical memory and the swap file is used, add more RAM to receive the biggest performance gain.

You may need more hard drive free space if your integration project includes either one of the following conditions:

- You are using a file-based data source or target connector, such as ASCII, Binary, Access, or DBase.
- You are running Map Designer on the same machine that drives a SQL RDBMS engine that also acts as source or target, such as SQL Server, Oracle, or Informix.
- **Use operating system tools.** Use your operating system built-in performance tools to find bottlenecks in hardware or software. For instance, Windows systems have the Performance Monitor utility that graphs various system metrics.
- **Set up a maintenance schedule.** Periodically delete unnecessary files, empty the Recycle Bin, compact databases, and defragment your hard disk.
- **Regular maintenance of the database.**
 - Perform disk optimization to reduce fragmentation.
 - Reorganize the tables and indexes to optimize the database.

Software

- **Check the version date.** Make sure that you have the latest available version that includes the most current patches. To do this, go to the main menu, and select **Help > About Map Designer > Product License**. Look at the value for Version.
- **After upgrading to a new version, do some testing.** Take time to plan and monitor processes, maps, and data to ensure there are no adverse effects from the upgrade. Read the release notes for Windows and other platform-specific information pertaining to the new release.
- **Do more investigating.** Use native or third party tools to test SQL statements and to check for connectivity performance issues.
- **Use best practices with your software applications.** If you experience performance problems accessing a SQL database, sometimes the root cause lies in the source or target application, not in Map Designer. Do some investigating by:
 - Checking SQL and RIFL scripts.
 - Finding unused objects, such as tables, and reclaim lost space.
 - Normalizing the database schema to avoid storing multiple copies of data.
 - Using cache memory. If you reuse the most recent data from the server while the application is running, it is faster to retrieve and store records in cache than to retrieve many individual rows.

Network Issues

- **Server speeds set the tone.** When you load source tables from a network, network speed can cause slow load times. Set up your system to allow records to be returned in batches, such as in record sets, or result sets, instead of returning the entire tables. Populating a lookup table requires that all records be retrieved, which slows performance. To improve performance, if data for lookup tables is never or rarely changed, make local copies of the lookup tables.
- **Network traffic during transformation time.** Try copying your source files and lookup tables to your local machine, and change the source paths accordingly. If this is not practical, try running transformations when there is less traffic on the server.
- **Other services running in background.** Ask IT administrators to turn off unnecessary services on their servers.

See Also

“Designing Faster Transformations and Processes”

“Running on the Command Line to Improve Performance”

“Using Process Designer to Run Maps on Multiple Threads”

“Comparing Engine Configurations”

Designing Faster Transformations and Processes

Transformations and processes are highly tunable. Careful use of certain functions, and limiting the amount of information reported on errors and warnings can greatly impact speed.

Design Tips

- **Set Commit Frequency.** This target property controls how often data is committed to the database. By default, it is zero. This means that data is committed at the end of a transformation. For large transformations, this is not practical because transaction logs grow quickly. Setting the Commit Frequency to a greater than zero value tells the connector to do a database commit after the specified number of records are inserted or updated in the table. This keeps the transaction log from getting too full, but limits the restartability of the transformation.
- **Large number of records to process.** To process a large number of records at a time, run the first transformation to your table in Replace Mode (if you are creating a new table). Then run the other transformations in Append Mode to add records to the table that you originally created. Use the source filters to limit the number of records to process in each transformation. If the problem persists, change the size to something smaller.
- **Dynamic SQL lookup tips.**
 - Place the queries for lookups into stored procedures. Stored procedures, unlike raw select statements, are compiled and optimized by the database engine, making them much more efficient.
 - Design a process with Process Designer and use a SQL step to execute the stored procedure to create a temporary table. Then use a transformation step to execute the transformation that reads the temp table created in the SQL step.
 - Add a clustered index to your lookup tables to see even faster run times. Searches in an indexed table are more efficient.
 - Declare and set DJImport objects in a BeforeTransformation event and not in a Target Field Expression cell. Also remember to destroy the object in an AfterTransformation Event.

- Consider using incore lookup functions that are much faster than the standard dynamic SQL lookups. Search for the keywords “ILookup”, “ILBuild”, “ILAddrow”, and “ILClear” in the online help.
- **Quicker updates with large source files.** When you update a large number of records in a SQL RDBMS, load all source data into a staging table, then perform an Update query directly inside the target database. You can reduce your update run-times by many minutes or hours by using the staging table method.
- **Record length can slow performance.** The longer the record, the longer the record takes to process. **Binary tip:** When parsing binary fields, try combining fields that are not used in the target into a single field.
- **Detect bottlenecks in a map or process by using the Engine Profiler.** You can troubleshoot files or tables up to 1000 records. For details, search for the keywords "Engine Profiling on the Command Line" in the online help.
- **Faster loads to database tables.** Bulk copy loads are the way to load very large tables. Remove indexes from the tables prior to loading, then reindex after loading. After updating the tables, make sure that the indexes appear in the tables.

Choosing a SQL Update Method

You may use one of two methods to improve performance.

Method 1 involves simply using the Update output mode alone: 39 min 30 sec Total Time *

Method 2 involves loading a SQL server with a staging table, and performing an Update with an ad-hoc SQL statement.

- 02 min 43 sec (Load Staging Table)

- 00 min 17 sec (Execute Update Query in SQL Server)

- 03 min 00 sec. Total Time

This method involves using a transformation to load the data into a temporary table on the database server. Use a SQL query to join the temporary table with the final target table, finding which key values match and which do not match. The query takes that merged view of the two tables and uses columns from one table to update columns in the other table. Because the tables are joined, the indexes on the server are put to better use and the update runs faster. Furthermore, fewer queries are sent between Map Designer and the server, so

network traffic is reduced and the server does not have to answer as many requests.



Note Transformation run times vary with different hardware and application scenarios.

Factors to Consider

The method you choose depends on the following factors:

- It is easier to write conditional logic and create a reject system in Method 1.
- Method 2 temporarily increases the database size and may result in more disk activity on the server than method 1.
- Method 2 requires some knowledge of SQL syntax.
- Method 2 is more flexible if you want to update staging records to indicate the results of an update.
- Method 2 provides more control over how the update is performed. For example, you can write the SQL statement to use a cursor, use a WHILE loop to scroll through records, or use a table join to merge records.

See Also

If you are using a multimode connector, you have the option of using the SQL Literal Property. Search for the keywords “SQL Literal Property for Multimode Connectors” in the online help.

“Enhancing Run-Time Performance”

“Using Process Designer to Run Maps on Multiple Threads”

“Comparing Engine Configurations”

“Running on the Command Line to Improve Performance”

Using Process Designer to Run Maps on Multiple Threads

The multithreaded engine designed for Process Designer can run more than one transformation in a process simultaneously, greatly reducing multiple transformation run times.

Parallel transformations are initialized and run at the same time, shortening the total run time. Process statistics, such as run time, and the number of records read, written, inserted, discarded, and rejected, are written to a .log file.

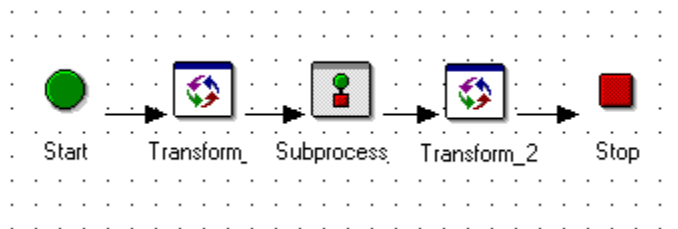
The multithreaded and multi-instance configurations can improve performance run times over the single threaded configuration.

- “Single Threaded Process”
- “Multithreaded Process”
- “Multi-Instance Threading”

Single Threaded Process

On a single-threaded engine, transformations and processes run sequentially. The single-threaded engine is also single-instance. An "instance" is an execution of an application and all threads associated with that execution. Transformations and processes run in the same thread within a single Data Integrator engine instance.

Information may be shared among transformations and processes as long as this information is applied in sequential order. This method is ideal when you do not require the fastest available performance.



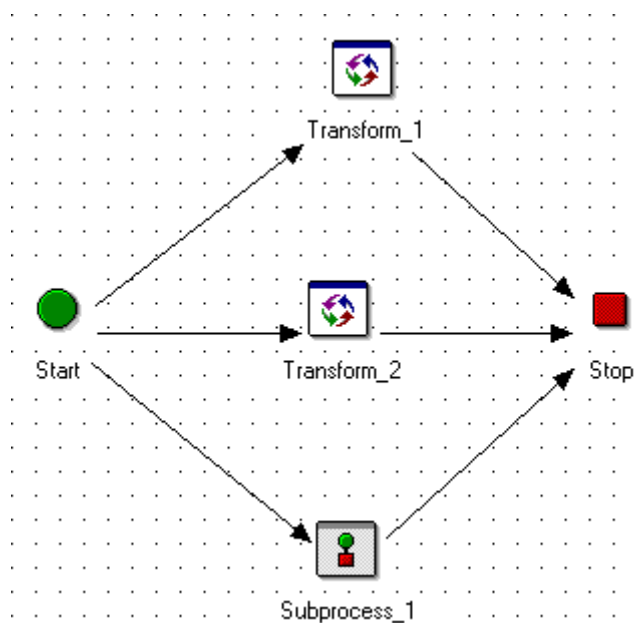
Multithreaded Process

On the multithreaded engine, transformations and processes to run concurrently and in on their own discrete thread. Your data integration solution can be developed with Process Designer or by using an integrated application through the API to start a single instance of Data Integrator engine.

Multiple threads within a single Data Integrator engine instance share the instance memory space, so they are also able to share information across threads. In cases where multiple transformations and processes have interdependencies that require simultaneous processing, shared information can have a positive effect on run-time performance. System resources can often be allocated more efficiently when operations run concurrently.

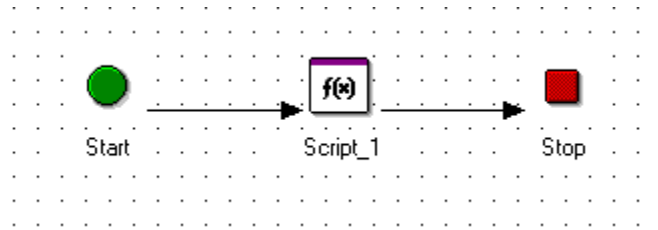
To take advantage of the multithreaded engine, do the following.

- Configure the number of execution threads in a process in the Process Properties window. In Process Properties, set *Max concurrent execution threads* to the number of parallel paths plus 1. So if you have three threads running in parallel, as in the process below, enter 4.
- Design a process with more than one transformation step. Then place all of the transformations parallel to one another between the Start and End steps.



Multi-Instance Threading

The multi-instance configuration calls more than one engine within the same process. You can write an expression in a Script step to call multiple engines.



The following RIFL script instantiates three integration engines concurrently:

```
Shell("djengine -Project_Execute "&_  
"Project_Root_Dir./MyRepository "&_  
"./MyRepository/SubProcess_1.ip.xml")  
Shell("djengine ./MyRepository/Transform_1.map.xml")  
Shell("djengine ./MyRepository/Transform_2.map.xml")
```

See Also

The *Samples Handbook* includes a number of working example processes. The handbook and sample files are available for download from the [Pervasive Software web site](#).

“Comparing Engine Configurations”

“Enhancing Run-Time Performance”

“Designing Faster Transformations and Processes”

“Running on the Command Line to Improve Performance”

“Troubleshooting Tips for Process Designer”

Managing SQL Sessions in Process Designer

When you add a transformation step in Process Designer and open an ODBC or SQL session connection, you may create more sessions than you actually need. Incorrectly managing SQL sessions can cause performance issues.

Once you learn how to manage sessions, you can determine which sessions you want to keep, and which to delete. You also need to know when to design single-threaded or multithreaded processes, and whether to create one session per transformation step or use one session for all transformations in the process.

➤ To manage sessions in Process Designer

- 1 In the left tree, select the **SQL Sessions** folder, then select **Manage Sessions**.
- 2 From the list, select a session name.
- 3 Review the properties of the session. Here, you can choose to edit the properties or delete the session.

Naming Sessions

When you first create a session, add a name for the session instead of accepting the numbered default name. The session name gives you valuable information when you are managing sessions and determining the purpose of the session. For instance, if you have three sessions that connect to an Oracle database, you could call them Oracle1, Oracle2 and Oracle3.

Designing for Performance

Using One Session or Many Sessions

One Session

If you are using a single-threaded sequential process to write to one table, you should use one session for all transformations. Since you are running each transformation step one after the other, it does not affect performance to reuse the same session.

Many Sessions

If your single or multithreaded process includes multiple transformations being written to multiple tables, consider using

separate sessions so that each session completes before another begins. For instance, you have three transformations (A, B, and C) being written to three different target tables. If you use the same session for all transformations, session A may still be open, and sessions B and C may try to open at the same time. Sessions B and C would have to wait for session A to release its memory before session B or C could open.

Reusing Sessions

We recommend that you reuse sessions by alternating the sessions when they are run in a series. For example, see the following example:

Series Example

Map A = Session 1

Map B = Session 2

Map C = Session 1

Map D = Session 2

and so on...

We recommend that you reuse sessions by separating the sessions when they are run in parallel. The following examples show scenarios with odd and even-numbered sessions.

Parallel Examples

Odd-Numbered Sessions

Map A = Session 1

Map B = Session 3

Map C = Session 1

Map D = Session 3

and so on...

Even-Numbered Sessions

Map AA = Session 2

Map BB = Session 4

Map CC = Session 2

Map DD = Session 4

and so on...

The examples above assume that the maps contain only one source or target database connection. If a map includes both a source and a target connection to a database, then the map requires two separate sessions as shown below:

Series Example for Source and Target

Map A: Source = session 1, Target = Session 2

Map B: Source = session 3, Target = Session 4

Map C: Source = session 1, Target = Session 2

Map D: Source = session 3, Target = Session 4

Using Session Properties to Best Advantage

TransactionIsolation Property

If you are reading and writing to the same table in different sessions, set the TransactionIsolation property to None. Since you want the table open to different sessions, set the property to None.

The default is Serializable, which means that one transformation step does not lock the table while other transformations attempt to write to the table.

Global Transaction Option

As a best practice, select Global Transaction in the SQL Session dialog to make your sessions transactional. The default setting is unselected. By setting the option, when the process encounters an error, any records that are set to be written are rolled back and not committed.

For example, a transformation step that transforms 100 records into a SQL database uses a transactional session. An error occurs on the 40th record that aborts the transformation. Since the session is transactional, no new records exist in the SQL database. If the session is not transactional, only 40 records out of 100 are written to the database table.

See Also

Search for the keywords “SQL Sessions” in the online help.

“Troubleshooting Tips for Process Designer”

Achieving Best Performance with Large Files or Tables

To transform large files or tables, use the method that provides the best performance.

Accessing Files or Tables **Filtering Smaller Set of Records**

During the test phase, you may want to filter the number of records that you process when connecting to a large file or table.

Instead of transforming all records, select a smaller value to limit the number of records for your test. Then when you are ready for production, you can transform the entire set of records. For more details, search for the keywords “Filtering Records” in the online help.

Use Iterators in Process Designer

An Iterator breaks large data sets into smaller parts that can be retrieved in order. Iterator components provide a mechanism to iterate over the elements of an object and to present each element as a separate message object. A typical use case of the iterator component is to break apart EDI documents to allow processing of document contents.

When using iterator components, pay particular attention to the following property settings:

- **Tablock Property:** Default is true. Set to false if you are writing to the same table in multiple transformations.
- **Identity Insert:** Default is false. Set to true if you know the value to insert in the primary key fields.

The following iterators are available in Process Designer:

General Iterators

JDBC WebRowSet Iterator

XPath Iterator

Health-Care Related Iterators

HL7 Batch Iterator

HL7 Message Segment Iterator

For more information on these iterators, search for the keyword “iterator” in the online help.

Use Mass Insert Connectors

Mass insert connectors provide a rapid way of inserting records into a database. The connector bypasses the transactional layer of the database and adds information directly to the storage tables. Mass insert connectors are designed to handle large database tables and provide the fastest performance. The following list includes some of the available connectors but is not all-inclusive:

- IBM DB2 UDB Mass Insert
- ODBC 3.x Mass Insert
- SQL Server 2000 Mass Insert
- SQL Server 2005 Mass Insert
- Sybase SQL Server Mass Insert
- Sybase Adaptive Server Mass Insert

For more details on mass insert connectors, search for the keywords “mass insert” in the online help.

Accessing Tables

Perform Faster Imports

Cursor support improves import performance. Since the cursor acts as a window to data, Map Designer does not need to read the entire result set into memory before using it. You can work with very large tables without running out of memory and without waiting for data to appear. For more details, search for the keyword "cursors" in the online help.

Faster Loads to Database Tables

Bulk copy connectors, such as SQL Server BCP and Sybase BCP are designed to write large amounts of data. BCP connectors provide best performance in simple transformations that do not contain lookups, numerous expressions, or event actions.

Bulk copy loads are an efficient method to load very large tables. You can remove indexes from tables prior to loading, then reindex the tables after loading. Once you update the tables, make sure that the indexes appear in the database.

BCP connectors create an output file that you can access in Process Designer using the following steps.

➤ **To use an output file in Process Designer**

- 1 Open Process Designer and create a new process.
- 2 Create an Application Step and point to the bulk load utility executable for your database application.
- 3 Point to the output file you created in Map Designer.

For more details, search for the keyword “BCP” in the online help.

Use DJImport and DJExport Objects

When using DJImport or DJExport objects, you should limit use of the `SQLStatement` property for table lookups to cases where performance is not a concern or lookups are infrequent. If lookups are performed using large tables or complex queries, it is recommended that queries be optimized through use of indexes and precompiled SQL statements. When performance is important and lookups are frequent, and the data set size is nominal and known, the use of incore lookup tables is recommended.

For more details, search for the keywords “DJImport” and “DJExport” in the online help.

See Also

“Choosing the Best Lookup Method”

“Designing Faster Transformations and Processes”

Comparing Engine Configurations

Review the following information to determine whether you should use a multithreaded or multi-instance configuration on your integration project.

Multithreaded and Multi-Instance

- Configure process properties to execute as single-threaded or multithreaded.
- Take advantage of multiple-CPU systems up to the limit allowed by the user license.

Multithreaded Advantages

- Shared memory address enables communication and sharing of information, among transformations and processes.
- Compared to multi-instance, system resources are conserved. Remember that each engine instance resides in its own memory space.

Multithreaded Disadvantages

- If a single transformation or process fails, an entire multithreaded execution may terminate.
- Concurrent threads that share common objects or information are at risk of fatal intercommunication conflicts.
- Performance may be compromised due to issues of logic, system resources, and the total number of concurrent threads.

Multi-Instance Advantages

- Memory space is not shared and transformations and processes are isolated from each other.
- Failure of any single transformation or process does not cause a failure of executions running in other engine instances.
- This approach, because of its component architecture, is more easily adapted to run across multiple systems.

Multi-Instance Disadvantages

- Programming complexity increases with interprocess communication.
- Interprocess communication, regardless of the method used, adds its own risk to the application.
- System resource consumption is greater than that of a multithreaded single-process approach.
- Communication between applications causes process context switches in the underlying operating system.
- Interprocess communication consumes more system resources than are required for multithreaded processes.
- Depending on available system resources, running a large number of instances concurrently can affect performance.

Further Discussion

You can also use a third-party scheduler such as Schtasks or CRON to execute the Data Integrator engine multiple times from the command line.

You may also run a map or process that uses the `Shell()` function to execute multiple instances concurrently. Search for the keywords “Shell Function” in the online help.

Business Case Scenarios

These examples demonstrate two transformations and two processes in single-threaded, multithreaded, and multi-instance configurations. In all the scenarios, Data Integrator engine resides on a four-processor server.

Descriptions of the Transformations and Processes

Transformation 1

COBOL to a relational database requiring two hours to load all data into eight tables with dependent keys.

Transformation 2

Delimited ASCII to a relational database requiring one hour to load data into the same tables.

Subprocess 1

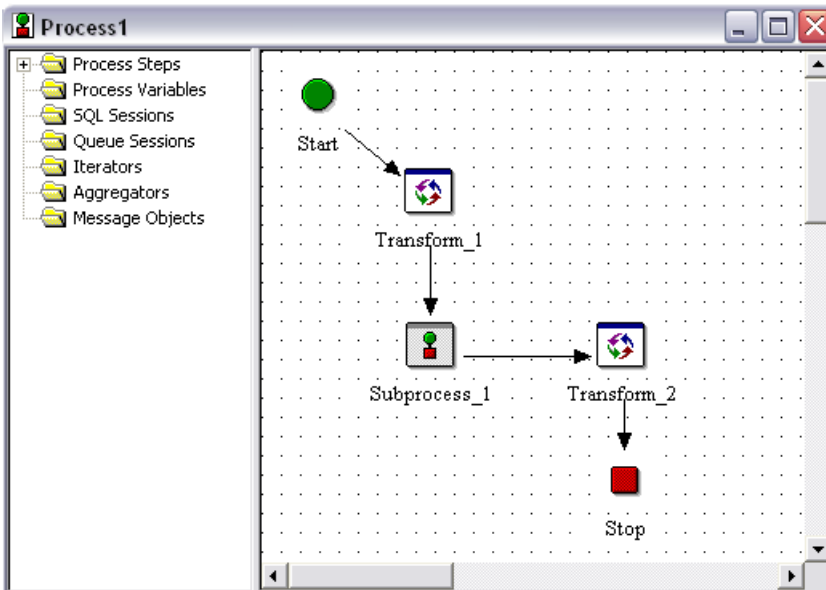
This process requires three maps to run in a specific sequence to load related, fixed ASCII files into the same relational database tables because of the dependent keys. It can be designed to run as a single-threaded or multithreaded configuration. This process requires three hours to complete.

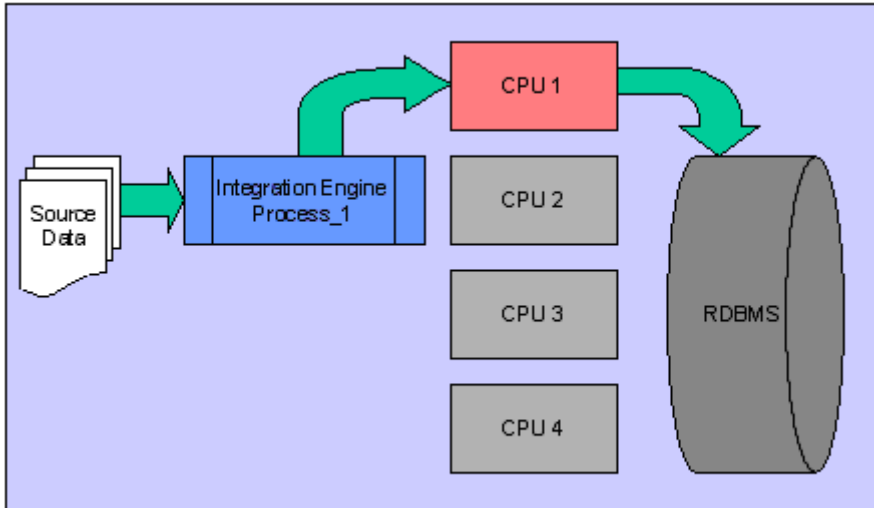
Process 1

The master process used to execute Transformation 1, Transformation 2 and Subprocess 1.

Scenario 1 - Single-Threaded

All files become available at the same time at the 6:00 p.m. close of business. The execution run must complete before 7:00 a.m. the following morning. The total processing time for these files is six hours. Single-threaded configuration should be able to run a sequential processing against a single CPU with ample margin to complete the work prior to the 7:00 a.m. deadline.

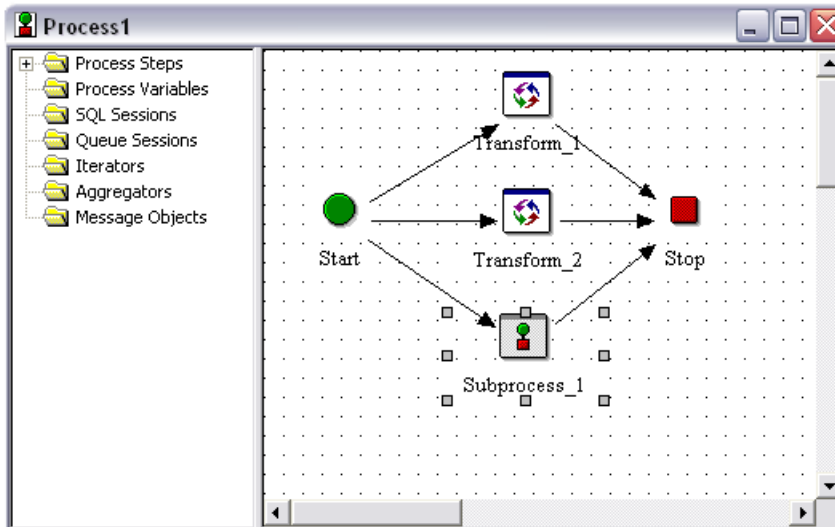


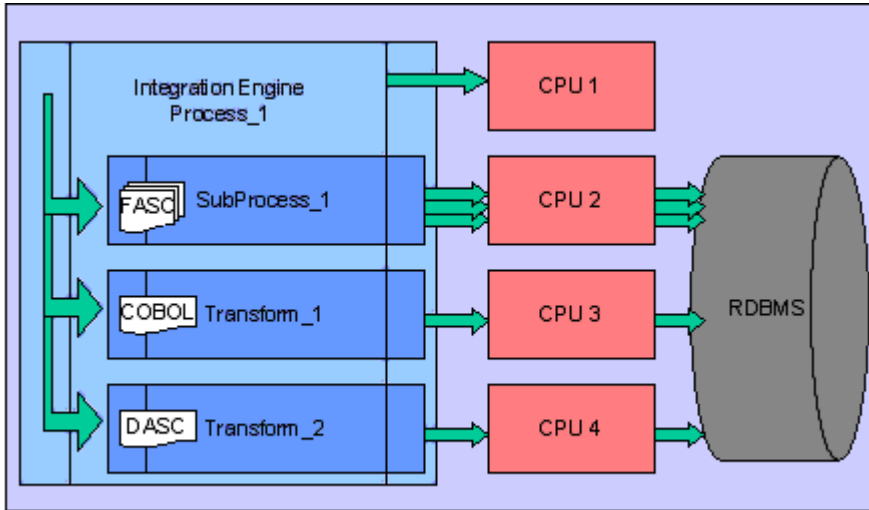


Single-Threaded Configuration, 4-CPU system

Scenario 2 - Multithreaded

All files are available at 9:00 p.m. and must be processed prior to a scheduled midnight reboot of all systems each night. This scenario allows three hours to complete the processing of all data files. In this case, Process1 runs the other transformations and processes in parallel to complete within the time allowed.

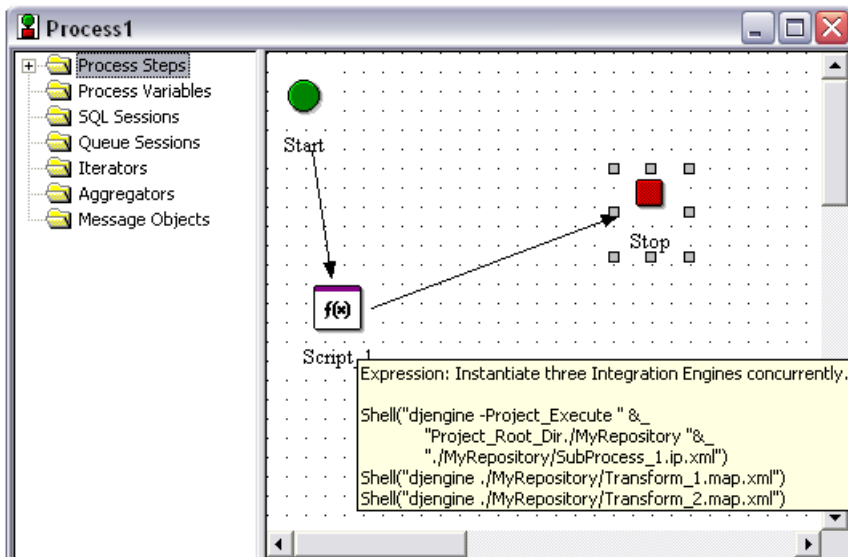


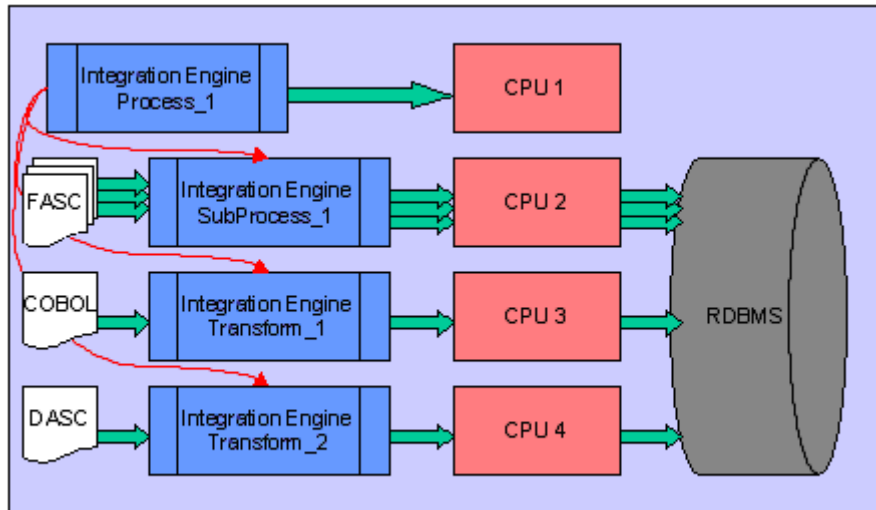


Multi-Threaded Configuration , 4-CPU system

Scenario 3 - Multi-Instance

For this example, the same business scenario as the multithreaded approach is processed as a multi-instance run. This is how the process appears in the Process Designer for a multi-instance approach:





Multi-Instance Configuration, 4-CPU system



Note The multithreaded and multi-instance diagrams show all threads from Subprocess_1 executed against only one CPU. The operating system, however, may balance these across different CPUs if the process were configured to run in multiple threads.

In our scenario, Subprocess_1 is designed to run its child steps in a specific sequence. Since each is executed sequentially, the run could be configured as single-threaded or multithreaded without an adverse effect on the outcome.

The differences between the multithreaded and multi-instance approaches are subtle but can be important. In the multithreaded approach, Process1 remains resident in memory until all steps have completed. If any single step aborts, the complete process fails. Error handling can capture and manage this possibility, but this adds overhead to the process.

In the multi-instance approach, the Shell() function issues the command and does not wait for a return. Process1 ends once the commands are made and frees the resources used by this instance.

The additional overhead involved with instantiating an Data Integrator engine for each Shell() function is balanced by reliability. If a single instance fails, it should not prevent the other instances from completing their work.

See Also

“Enhancing Run-Time Performance”

“Designing Faster Transformations and Processes”

“Using Process Designer to Run Maps on Multiple Threads”

“Running on the Command Line to Improve Performance”

Running on the Command Line to Improve Performance

If you want to achieve the fastest processing times, use Data Integrator engine instead of running processes or maps in the designer tools. Running large transformations on the command line eliminates user interface delays.

Troubleshooting command line issues differs slightly from issues arising when using the design tools. The following checklist may be helpful.

Command Line Troubleshooting

- Review *Error Codes and Messages*.
- Check the command line syntax.
- Verify the tf.xml file is used for maps.
- Be sure map or process is specified last on the command line.
- Check the spelling of file names, variable names, and so on.
- Verify that the license has not expired by using `djengine -V`.
- Confirm the appropriate version is installed.
- Does the process or map run from the design tools?
- Are your environment variables set up correctly?
- Try a backup or previous copy of your file.
- Are you using the correct case? The following can be case-sensitive:
 - ◆ Macro names
 - ◆ Commands on Unix platforms
 - ◆ Switches (for example, `-V` or `-v`)
- Does it run on one platform and not on another?
- Check your file path slashes.
 - ◆ Windows paths should use backslashes.
 - ◆ Unix and web paths should use forward slashes.
- Use the `-pe` option for processes (`.ip.xml` or `.djar`). Make sure this option appears immediately after the `djengine` command.
- If macros are used, verify that the macro file is listed in the `cosmos.ini` or that the `-mf` option is being used.

See Also

To run transformations on the command line, search for the keywords “To Run a Transformation” in the help.

To run processes on the command line, search for the keywords “To Run a Process” in the help.

“Enhancing Run-Time Performance”

“Designing Faster Transformations and Processes”

“Using Process Designer to Run Maps on Multiple Threads”

“Comparing Engine Configurations”

Error Handling

chapter

3

Tips for Handling Errors in Transformations and Processes

This section covers the following topics:

- “Error Handling and Trapping”
- “Error Handling Tips for Processes”
- “Managing Log Files”
- “Using the Error Log as a Troubleshooting Tool”

Error Handling and Trapping

When you handle errors properly, production maps and processes can continue to execute. Error handling is important for the following reasons:

- If one bad record or one bad file causes an error, you probably want to continue processing other records and files and not have your entire process stop.
- You may not discover what happened until the next time someone checks the log file or finds files are not where they are expected to be.

Use the following topics as a guide when designing your transformations.

- “Setting Map Execution Properties”
- “Selecting Map Error Logging Properties”
- “Using OnError Events”
- “Catching Run-Time Errors”

Setting Map Execution Properties

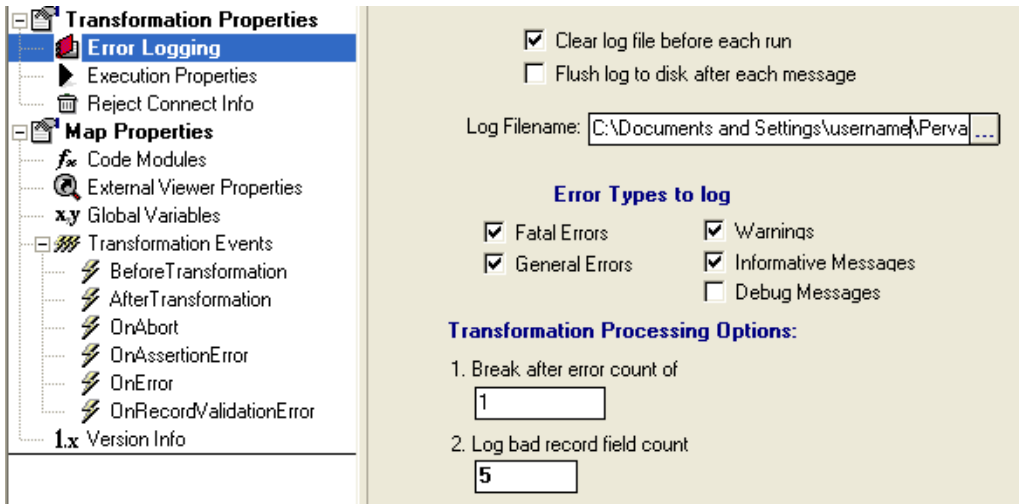
These properties determine whether the condition is considered an error, a warning, or is ignored. You may need to change the default property settings to those shown below.

Execution Properties	
Property	Value
Overflow Handling	Ignore
Truncation Handling	Treat as Warning
Null Handling	Ignore
Source Schema Mismatch	Treat as Error
Target Schema Mismatch	Use Map Schema

Use Execution Properties from Preferences (Runtime tab)

Selecting Map Error Logging Properties

To determine what to write to the log file, set the following options in the Transformation and Map Properties window.



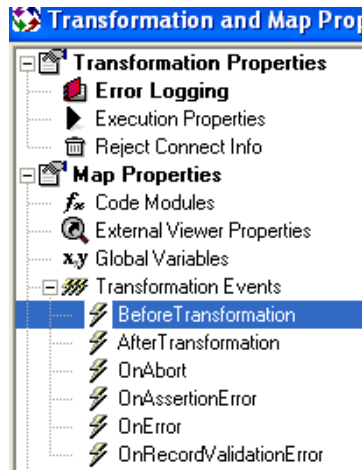
Using OnError Events

Use these steps to set up error events.

- 1 Determine which events to use.
- 2 Build the appropriate actions.
- 3 Write a message to the log file.
- 4 Send the record to a reject file or table
- 5 Use the Resume action to continue the transformation or process.

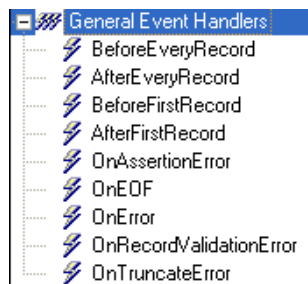
Determining Which Events to Use

Capture errors with specific or general OnError events.



BeforeTransformation Events

- OnAbort
- OnAssertionError
- OnError
- OnRecordValidationError



General Event Handlers—Source

Use the following event handlers for source files and tables.

- OnAssertionError
- OnEOF
- OnError
- OnRecordValidationError
- OnTruncateError

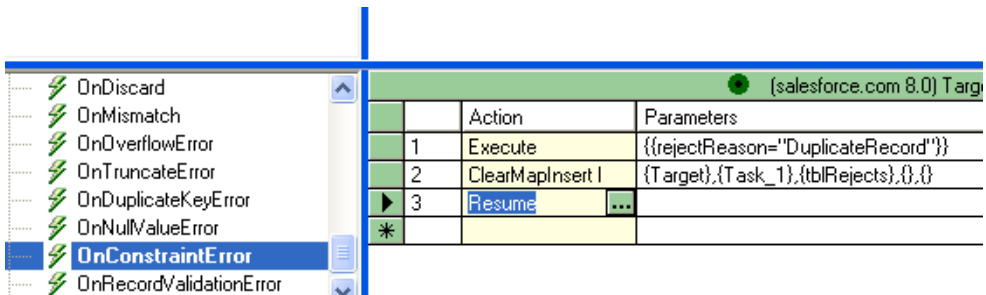
General Event Handlers—Target

Use the following event handlers for target files and tables.

- ⚡ OnAbort
- ⚡ OnAssertionError
- ⚡ OnReject
- ⚡ OnDiscard
- ⚡ OnMismatch
- ⚡ OnOverflowError
- ⚡ OnTruncateError
- ⚡ OnDuplicateKeyError
- ⚡ OnNullValueError
- ⚡ OnConstraintError
- ⚡ OnRecordValidationError
- ⚡ OnError

Building the Event Handler Actions

Once you select an event handler, choose the appropriate event actions. In the example below, the OnConstraintError event is triggered when a duplicate record is encountered. After the ClearMapInsertRecord action rejects the duplicate record to a reject table, the transformation resumes.



Catching Run-Time Errors

Use the On Error GoTo statement to catch run-time errors.

```
Function StateCodeLookup(state_code)
'Define the error handler:
On Error Goto err_StateCodeLookup
'Declare myimport as Static so that you do not lose the
instance data each time the function is executed:
Static myimport As DJImport
'Set the Connector and connection info:
```

```
set myimport = new djimport "Oracle 10g"
myimport.ConnectionString="database=default;username=scott
;password=tiger"
'Attempt to find the state code in the lookup table:
myimport.SQLStatement = "Select * from lookup where
STATECODE='" & state_code & "'"
StateCodeLookup = myimport.fields("STATECODE")
Return
'Return a default value if the entry in the lookup table
cannot be found, or you can fire an Input box or call
another function:
err_StateCodeLookup:
if err.number = 119 then
StateCodeLookup = "Unknown"
else
'Trap all other errors and write the error to the log
file:
logMessage("Error", ">>> " & "Error " & err.number & " -
- " & err.description & " occurred during lookup of value
" & state_code)
end if
Resume
End Function
```

See Also

“Error Handling Tips for Processes”

“Managing Log Files”

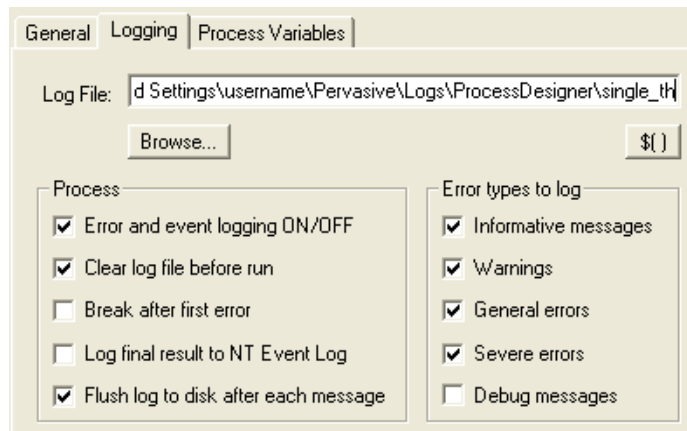
“Using the Error Log as a Troubleshooting Tool”

Error Handling Tips for Processes

To most effectively design processes for your own integration projects, you should change some of the error handling default settings in Process Designer.

Global Process Level

Default behavior within any process is to abort when any error is encountered, fatal or not. This behavior can be changed in Process Properties.



As a best practice, change the logging settings as follows:

- Select **Clear log file before run** and **Flush log to disk after each message** and clear **Break after first error**.
 - **Clear log file before run** ensures that only information from a single execution is stored in any particular log file. If prior execution log history is desired, archive individual log files instead of appending them to each other.
 - **Flush log to disk after each message** ensures that all log messages generated during execution are written to a log file. If this checkbox is not selected, the system determines when to write out log messages to the log file. If a fatal error is encountered between system designated log flushes, then any messages not written to disk are lost. As disk access is

often slower than processing routines, this may have an effect on performance. Alternatively, use the FlushLog function in strategic areas of code to control when log messages are written to disk.

- **Break after first error** is checked by default and causes the process to abort immediately when any error is encountered. This property should be unchecked and used in conjunction with the “Ignore error” property in each process step. Default behavior within any process step is to abort when any error is encountered, whether it is fatal or not. Select “Ignore error” in process steps as shown below.

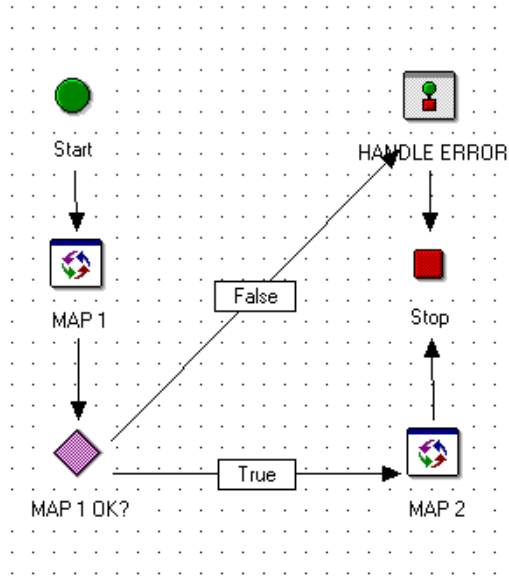
The screenshot shows a configuration dialog for a process step. The fields are as follows:

- Name: Queue_1
- Description: MSMQ queue session
- Ignore error
- Queue session: QSession-1 [MSMQ 1.0.0] (with a Sessions... button)
- Action: GetMessage
- Parameters table:

Property	Value
Queue Name	
Message Name	Queue_1_Output

Process Step Evaluation

In most instances, successive process steps are dependent on the execution of previous steps. For example, a second or third map within a process may be dependent on the target of a previous map. A transformation may use the target of a previous transformation as its source. In such cases, it is good practice to evaluate that the previous transformation executed properly prior to continuing. This is done with a decision step.



In the this example, the decision step MAP 1 OK? evaluates the following RIFL code, which checks that the previous map executed correctly and without error:

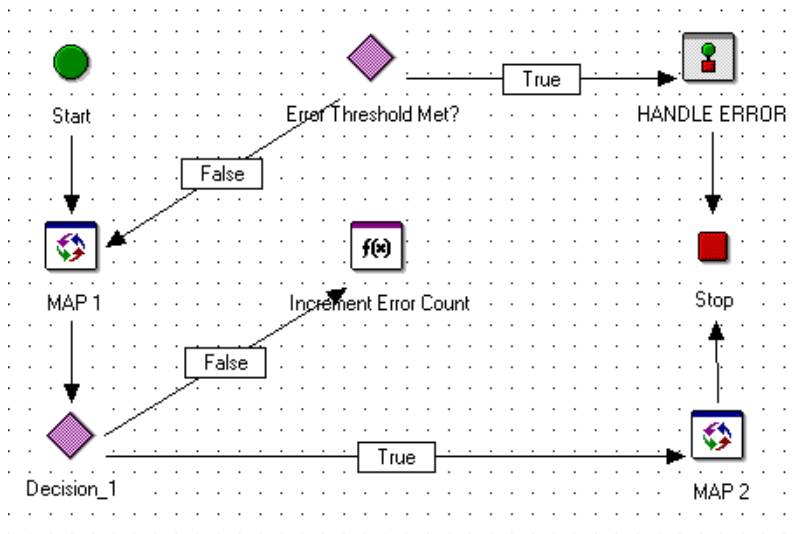
```
Project("MAP 1").status <> "Not Executed" AND
Project("MAP 1").ReturnCode == 0
```

These can be automatically built with the Step Result Wizard within any decision step. You can also test for other indicators that may be meaningful for deciding whether or not to continue. For instance, if the MAP 1 target is a flat file or djmessage type, check to see whether either contains data using the following tests:

- `FileLen("someFile.txt") > 0`
- `Len(someMsg.body) > 0`

The HANDLE ERROR step is shown as a subprocess step, but it could also be a scripting step that calls an external code module. You can create and maintain a single subprocess to use throughout your enterprise to handle these types of errors in the same way each time encountered.

If a particular map or step fails the first time, you may need to make several attempts by building a finite loop. In this example, MAP 1 attempts to execute until it reaches its Error Threshold.



RIFL Scripting and Function Steps

Run-time errors can and should be handled within RIFL code any place where RIFL code can exist, including function steps, target field expressions, and execute actions. The mechanism used for this is the On Error Goto syntax.

Example

```

On Error Goto someHandler
*
*Normal execution code goes here
*
'If execution is OK, then drop out of module
Return
'If error encountered goto here
someHandler:
'Use the Err object to get information about the error.
LogMessage("ERROR", "***** ERROR *****")
LogMessage("ERROR", "Error Number: " & Err.Number)
LogMessage("ERROR", "Description: " & Err.Description)
LogMessage("ERROR", "Line of Code: " & Err.Line)
'Check to see if the error is detrimental.
If thisError is OK <PSEUDOCODE> then
    Resume Next
  
```

```

Else
    LogMessage("ERROR", "Fatal Error - Aborting
Process...")
End If

```

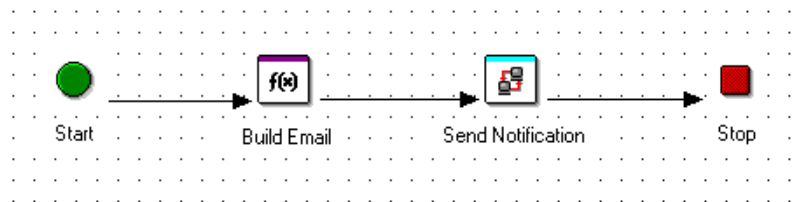
E-mail Notification

To handle error notification, use one or both of the following methods.

- When a detrimental error is handled, e-mail the log file to the process administrator.
- Periodically parse log files for error messages and transform the data into a monitored database.

The E-mail Invoker can send mail through both SMTP and IMAP servers.

You can build a simple subprocess to handle error notification. Use a Script step to contain the e-mail message and an E-mail Invoker step to send the notification.



In the Build E-mail step, use the following RIFL code.

```

Set logfile = new DJMessage "logfile"
Set email = new DJMessage "email"
Dim logfileBody
logfileBody = MacroValue("LOG_FILE")
'Build the e-mail message and add To, From, and Subject
email.Body = "Execution log for " &
MacroValue("DISTRIBUTOR")
'Read the log file and add as an e-mail attachment
'The FlushLog Function ensures that you capture recent
log messages.
FlushLog()
logfile.Body = FileRead(logfileBody)
Dim slash

```

```
slash = CharCount("\", logFileBody)
logFile.Properties("FileName") = Parse(slash,
logFileBody, "\")
email.addAttachment(logFile)
```

Error notifications are self-contained in a subprocess and are independent of the master process, the error that occurred, and the error-handling routines in the master process. Once the notification is sent, the administrator receives the log file to assist with troubleshooting.

See Also

“Error Handling and Trapping”

“Managing Log Files”

“Using the Error Log as a Troubleshooting Tool”

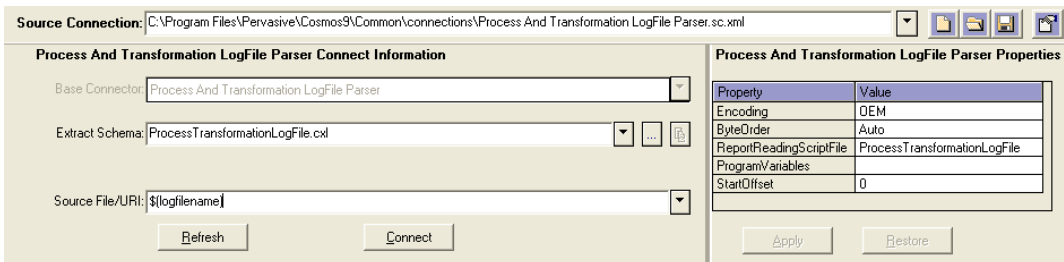
Managing Log Files

Successful log file management involves automated log file monitoring and automated log file archival.

Automated Log File Monitoring

Fatal errors are usually not captured by normal error handling and notification schemes but are often captured by the logging mechanism. For instance, you can create a separate process which periodically reviews current log files for fatal errors.

The Process and Transformation Log File Parser connector can be used to traverse current process logs for fatal errors. You can collect errors and send the errors in a notification e-mail to the process administrator. For details on the connector, search for the keywords “Process and Transformation Log File Parser” in the online help.



Parsing the log file

Use the Process and Transformation Log File Parser connector to pull information from the log file.

The screenshot shows a software interface with a menu bar (File, Connection, View, Tools, Add-Ins, Help) and a toolbar. Below the toolbar are two tabs: 'Source Connection' and 'Target Connection'. The main area is divided into two panes. The left pane, titled 'Record Types', lists various record types such as 'Project_Execution_Stats', 'XSLTTransformer_Step_End', 'SubProject_Step_End', 'Application_Step_End', 'Decision_Step_End', 'Transformer_Step_End', 'Skipped', 'DTS_Step_End', 'Invoker_Session', 'Aggregator_Session', 'Iterator_Session', 'Transformer_Session', 'Information_Messages', 'Warning_Messages', 'Prepare_Map', 'Transformation_Last_Error_Final', 'Debug_Messages', 'Queue_Step_End', 'Scripting_Step_End', 'Iterator_Step_End', 'Aggregator_Step_End', 'Transformation_Step_End', 'Start_Step_End', 'Stop_Step_End', 'DocValidator_Step_End', 'Invoker_Step_End', 'SQL_Step', 'SQL_Session', 'Queue_Session', 'Transformation_Totals', 'Transformation_Totals Event Handlers', 'Transformation_Totals Fields', and 'Transformation_Totals Rules'. The right pane, titled '(Process And Transformation LogFile Parser) Source Record Type: Transformation_Totals', displays a table with the following columns: Source Field Name, Description, Type, Size, and Contents(Rec 1). The table contains 27 rows of data, including fields like 'Total_records_Buffered', 'Execution_Time', 'Execution_Begin_Time', 'Execution_End_Status', 'Map', 'Error_Description', 'Prepare_Map_Time', 'Total_Error_Count', 'Total_records_Deleted', 'Total_records_Discarded', 'Total_Records_Inserted', 'Records_Read', 'Total_Rejected', 'Total_records_Updated', 'Total_Records_Written', 'Total_Records_Rejected', 'Map_Version', 'Execution_End_Time', 'Error_Time', 'Execution_Init_Time', 'Transformation_Last_Error', 'Error_Number', 'Organization', 'Owner', 'Product', 'Serial_Number', and 'Build_Version'.

Source Field Name	Description	Type	Size	Contents(Rec 1)
* <all fields>				
1 Total_records_Buffered		Text	1	<null>
2 Execution_Time		Text	14	<null>
3 Execution_Begin_Time		Text	20	<null>
4 Execution_End_Status		Text	23	<null>
5 Map		Text	12	<null>
6 Error_Description		Text	25	<null>
7 Prepare_Map_Time		Text	20	<null>
8 Total_Error_Count		Text	1	<null>
9 Total_records_Deleted		Text	1	<null>
10 Total_records_Discarded		Text	1	<null>
11 Total_Records_Inserted		Text	3	<null>
12 Records_Read		Text	3	<null>
13 Total_Rejected		Text	1	<null>
14 Total_records_Updated		Text	1	<null>
15 Total_Records_Written		Text	3	<null>
16 Total_Records_Rejected		Text	1	<null>
17 Map_Version		Text	8	<null>
18 Execution_End_Time		Text	20	<null>
19 Error_Time		Text	20	<null>
20 Execution_Init_Time		Text	20	<null>
21 Transformation_Last_Error		Text	1	<null>
22 Error_Number		Text	1	<null>
23 Organization		Text	25	<null>
24 Owner		Text	25	<null>
25 Product		Text	1	<null>
26 Serial_Number		Text	12	<null>
27 Build_Version		Text	9	<null>

Automated Log File Archival

You should maintain log files for at least one month. Include RIFL code in a final function step or in the stop step of a process. Copy the current log file to an archival location with a naming convention used to control the archival period. For example, a single line of RIFL code can set up a one-month archive:

```
FileCopy (MacroValue ("LOG_FILE")
MacroExpand ("$(LOGS)Archive\ToBeNamed" & Day(Now()) & ".log"))
```

Any log files more than one month old are overwritten. To create your own custom solution, modify the archive period and the copy frequency.

See Also

“Error Handling and Trapping”

“Error Handling Tips for Processes”

“Using the Error Log as a Troubleshooting Tool”

Using the Error Log as a Troubleshooting Tool

The error and event log records messages to help you debug transformations and processes. You can make a few adjustments to get more information in the log file.

Customize error logging settings. You may not want to use the default settings for fatal errors and warnings. You can change the settings for all transformations, or for a single transformation. For details, search for the keywords default and preferences in the online help.

Use the TraceOn and TraceOff Actions. These actions turns on and off error tracing so that details about each record transformation are written in the error and event log file.

In your transformation designs, do the following to use TraceOn and TraceOff actions:

- Select **View > Transformation and Map Properties > Error Logging** and make the following changes to the defaults:
 - Turn **Debug messages** on.
 - Select **Flush Log to Disk After Each Message**.
- In a BeforeTransformation event, set the **Trace On** action.
- In an AfterTransformation event, set the **Trace Off** action.

For examples using the TraceOn and TraceOff actions, search for the keywords TraceOn, TraceOff, and debug in the online help.

Use the LogMessage Action. This action allows you to send a custom message to the log file. For instance, you can pair this action with the OnTruncation error event and log a message when data truncation occurs. For an example, search for the keyword LogMessage in the online help.

Use the LogTargetRecord Action. This action writes current target record field values into the error and event log file. You may want to analyze target record field values on the fly in the log file. For details, search for the keyword LogTargetRecord in the online help.

Clear the error log. Periodically open the log and select the Clear Log option. You can also have Map Designer automatically clear the log prior to running a transformation. Choose one of the following options:

- To clear the error log each time a transformation is run, select **View > Preferences** from the menu. Check the **Clear Log File Before Each Run** option.
- To select the option on specific transformations, select **Transformation and Map Properties**, then select **Error Logging** from Options. Check the **Clear Log File Before Each Run** option.

See Also

“Error Handling and Trapping”

“Error Handling Tips for Processes”

“Managing Log Files”

Appendix

chapter

A

Supplemental Information

This appendix provides information important to selected users but is not essential to running the integration products.

It includes the following topics:

- “Automating Data Integrator Engine”
- “Choosing the Best Lookup Method”
- “Generating Test Data Using the Null Connector”
- “Comparing Unix and Windows Deployments”
- “Employing Code Modules and User-Defined Functions for Reusability”
- “Troubleshooting Tips for Process Designer”

Automating Data Integrator Engine

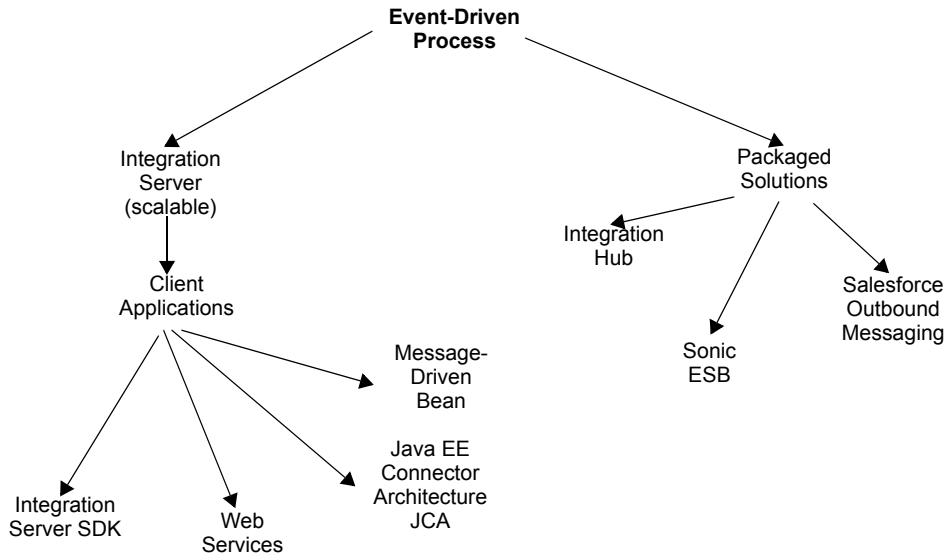
Latency, or time waiting for a response to a need, is an important consideration when planning to automate the engine. After you have decided whether to use an event-driven or a batch approach, you can determine the rest of the work flow.

- **Running event-driven processes**

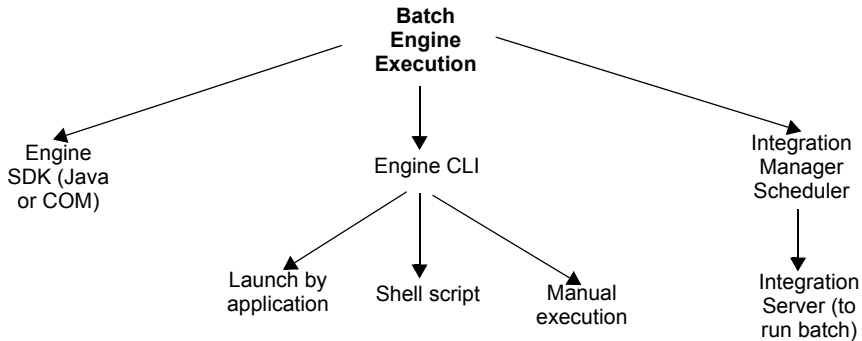
Use Integration Server or a packaged solution to respond to requests as they are received. This method involves low latency, or little to no delay between the request and each engine run. See the diagram for methods you can use.

- **Running engine jobs in batch mode**

Run batches of files at scheduled times. This method involves high latency, or a time delay between engine runs. See the batch execution diagram for methods you can use.



If you choose batch engine execution, you can use an engine SDK, the engine command line interface (CLI), or the scheduler in Integration Manager.



When choosing between the engine CLI or SDK, remember that memory resource usage is different. The advantages and disadvantages of each approach are detailed below.

Engine SDK	Engine Command Line Interface
<ul style="list-style-type: none"> • More fine-grained control • Tightly coupled • Share variables between maps and processes (Shared Expression Context) • Engine resources compete for application resources • Better for small requests • Reuse maps for one startup cost 	<ul style="list-style-type: none"> • Out-of-process • Loosely coupled • Better resource management • More overhead • One start, one request • Better for large units of work • More crash resistant

User Scenarios These scenarios explain why companies A, B, and C chose their engine automation method.

Batch Engine Execution

Company A is a bank that needs to process bank transactions hourly. They use Integration Manager to schedule the engine to run each hour and Integration Server to run file batches.

Running Event-Driven Processes

Company B is a doctor's office that wants to process data as soon as it is received. They use the Salesforce outbound messaging component to listen for data, launch a transformation, and create Excel spreadsheets from the source data.

Packaged Solution Calls Web Service

Company C is a software company that wants to use an event-driven process to listen for files. They purchase a packaged solution that uses web services. Using Integration Server, they call the web service to process customer online requests.

See Also

Integration Server User's Guide

Integration Server SDK

Integration Manager Installation and Configuration Guide

Search for the following keywords in the documentation: “salesforce outbound messaging”, “Sonic ESB”, “scheduling transformations and processes”, “scheduling transformations and processes as tasks”, “AT scheduling on Windows”, “Cron-It! on Windows”, “Cron on Unix”, “.bat file”, “Java SDK” or “COM SDK”.

Choosing the Best Lookup Method

Lookup methods are important to you if you want to do any of the following:

- Data merging
- Data validation or cleansing
- Retrieve batch numbers or identity values
- Data aggregation

There are currently five lookup methods:

- “Inline or Search and Replace”
- “Static Flat File”
- “SQL Pass-Through”
- “Dynamic SQL”
- “Incore Table Lookup”

Inline or Search and Replace

Consider the Inline method to do any of the following:

- Fuzzy pattern matching
- Character translations
- String substitutions
- Simple data validation

Inline – Methods

Sub()

- Regular expression (pattern) based
- Replaces only first instance

StrReplace

- Literal-based
- Supports decimal/hex characters
- Can be throttled (number of occurrences to replace)

GSub()

- Regular expression (pattern) based
- Replaces all occurrences in a string

Transliterate()

- Literal-based
- Character by character substitution
- Mainly used for character set mapping, control character stripping/substitution

Very Simple to Use

- RIFL expressions allow very fuzzy pattern matching
- Must know one to four functions and a few flow control statements

Very Portable

- Map can contain all lookup logic
- Can encapsulate with user-defined function/external code module
Avoids repeating expressions when used against several fields
- Adds modularity (external code module can be updated independently of transformation)
- Can be entirely contained in the transformation specification file

Very Static

- Search values/patterns and replacement values are hard-coded, so changes to key values, patterns, or logic require editing expression code
- Must anticipate what values may occur in source

Questionable Performance

- Limited scalability – does not scale well past a few pattern searches or translations
- Depends upon the function used; certain expressions can slow performance

Static Flat File

Consider the Static Flat File method to do any of the following:

- Lookups of data passed between sites, partners, or vendor to client
- Lookups of data that exists in non-SQL format
- Preexisting ASCII lookup files

Static Flat File – Methods

Lookup Function

- Lookup file must have two columns — one key, one corresponding value

Xlate Function

- Lookup file must have two columns — one key, one corresponding value
- Supports strings, decimal values, hex values

TLookup Function

- Lookup file can have multiple columns — one key, n number of corresponding values

Very Simple to Use

- Must know one to two lookup functions
- May need to know parsing functions
- May need to know how to create a lookup file

Fairly Portable

- Can move site to site, vendor to vendor
- Lookup tables are external to transformation and can be independently updated
- May need to change lookup table paths

Fairly Static

- Lookup data must be in flat file (Delimited ASCII) format
- Lookup table must exist when transformation loads
- Can construct lookup table immediately beforehand using another transformation
- Data is static during transformation run time

Performance

- Degrades linearly as table size increases (key length, number of rows)
- Lookup table is indexed on the fly when transformation is loaded
- Map Designer will load table and index into memory if possible

Considerations

- Must anticipate what lookup data is needed
- Maximum line length restrictions (4K)
- Maximum key length restrictions (31 chars)
- No field delimiters
 - Field separator must be unique (does not appear in lookup data)
- Record separator must be CR or NL or CR/NL
- Last line must have a record separator!
- Can use relative paths to ease migration between machines/sites

SQL Pass-Through

Consider the SQL Pass-Through method to do any of the following:

- Data aggregation
- Complex sorting
- Statistical analyses
- Source and lookup data already existing in same query-aware data store

SQL Pass-Through – Methods

- Ad-hoc Queries vs. Views versus Stored Procedures

Ad-hoc queries are the most flexible, but also the slowest:

- Views allow the DB server to parse and develop an optimized execution plan **once** for queries requested multiple times
- Stored procedures have the benefits of views, but also allow parameters to be passed and allow other functions (inserts/updates) to be scripted

Map Designer

- Views (connect just like tables)
- SQL Pass-Through
- SQL File

Integration Engine

- Query statement / file can dynamically changed (overridden) via SDK API and command-line parameters

Moderate Complexity

- Must understand query language (such as dBASE, SQL)

Limited Portability

- Source data and lookup data must reside in the same data store
- Data store must support queries
- Can preload data store with lookup table using an additional transformation
- Only one source connection string can be edited via Map Designer, Engine command line override, or Engine API
- Table structures (column names and data types) must match between sites/databases
- Switching source connectors may require editing of query statement
- Can use stored procedures to increase portability

Somewhat Dynamic

- Query gets processed at the beginning of the transformation
- Major database servers allow dynamic record sets
- Can use ChangeSource Action to reload
- Data is as current as when source was last connected
- Major database servers support dynamic cursors, which reflect Updates after initial fetch

Unbeatable Performance

- Best overall performance over all other methods (any size lookup table)
- Intelligent use of indexing and query tuning can yield better results
- Indexes, clustering, off-loadable to separate machines create massively scalable options
- DB Replication Support offers additional (heterogeneous) possibilities
- Must be able to express correlation between source & lookup data using query syntax

Dynamic SQL

Consider the Dynamic SQL method to do any of the following:

- Retrieval of sequence/batch numbers and identity values
- Interrogate target
- Duplicate key values must be handled
- Very rapidly changing lookup data

- Massive lookup tables (millions of rows and many columns)

Dynamic SQL – Methods

Using Object Variables

- Declare object variable
- Instantiate object variable
- Set or read properties
- Destroy object reference

DJImport Object

TypeName (Read Only) Value (Read Only)

ConnectionString (Write Only) SQLStatement (Write Only)

FieldCount (Read Only) RecordNumber (Write Only)

Fairly Complex

- Must understand query language (dBASE, SQL)
- Must know how to build connection strings
- Should have working knowledge of object variables
- Should understand expression error handling
- Must understand how to programmatically connect to data store
- May need to know some flow control statements

Limited Portability

- Connection strings must be edited in Map Designer
- Changes in table structures (column names and data types) must match between sites/databases
- Switching source connectors may required editing of query statements
- Can use stored procedures/views to increase portability

Very Dynamic

- User controls when and how often queries are processed
- No need to anticipate what lookup data is required
- Allows lookup data to be heterogeneous to main source & target
- Data is as current as the last time the SQL Statement property was set

Big Performance Hit

- Query gets executed multiple times, incurring transaction and network costs
- Overtakes other options (except SQL Passthrough) on massively large lookup tables, or when size of lookup data size far overshadows size of source data
- Dynamic SQL query gets executed repeatedly, therefore there are transaction / network costs.
- The expression and query syntax can greatly affect performance.
- This method can become practical, however, if you have very large (million+) lookup tables.

Notes:

- Non-SQL unaware users can easily impair performance by making mistakes.
- Lookup key/parameters can be processed with expressions before being passed to data store. This allows a fuzzier correlation between source and lookup data.
- Great for interrogating target or getting batch/identity/sequence values.

Incore Table Lookup

Consider the Incore Table Lookup method to do any of the following:

- Lookups of data that exists in a query-aware data store
- Heterogeneous source and lookup data stores

Incore Table – Methods

Using Incore Lookup

- Create and connect DJImport object
- Build incore lookup table
- Query incore lookup table
- Clear incore lookup table
- Destroy DJImport object reference

New Functions

ILBuild Function – Creates or appends to incore lookup table

ILAddRow Function – Creates / appends to incore lookup table (one row at a time)

ILClear Function – Destroys an existing incore lookup table

ILookup Function – Queries an existing incore lookup table

Fairly Complex

- Must understand query language (dBASE, SQL)
- Must understand object variables
- Must understand how to programmatically connect to data store (build connection strings)
- Must understand expression error handling
- Must understand flow control statements

Limited Portability

- Must edit connection string in Map Designer
- Changes in table structures may require changes to expressions
- Switching source connectors may require editing of query statements
- Can use stored procedures/views to increase portability

Somewhat Dynamic

- User can control when table gets built, cleared, and refreshed
- Table can be rebuilt several times throughout a transformation
- Can add rows one at a time to an incore table
- Data is as current as the last time the incore table was built/rebuilt

Performance

- Lookup data is indexed and loaded into RAM directly from data source
- Balances control and dynamism of Dynamic SQL with performance of Static Flat File Lookups
- Majority of time is spent building and indexing table in memory
- Memory pressure can become a problem with massively large lookup tables

Generating Test Data Using the Null Connector

Map Designer includes a feature that allows you to create test data. With the null connector, you can design a transformation even when you do not have access to the source data table.

The primary purpose of this feature is to allow you to generate output data when there is no input data. It can also be used to do the following:

- Create partial transformations. This is helpful when you repeatedly transform varied source data to a single target format.
- Design a transformation that will be ported to another system.

➤ **To use the null connector**

- 1 Select **Null Connector** as the source connection.
- 2 Select **Null Connector** as the target connection to create a null target without any records.
- 3 Next, you can map the source to the target file and run the transformation. A null target with 100 records is created.

Comparing Unix and Windows Deployments

When you deploy Data Integrator on Unix operating systems, here are some design considerations to keep in mind.

Limitations

- The lack of integration design tools makes development and testing less reliable. You must design on Windows and deploy the specification files to Unix.
- Some connectors have limitations. For example, Microsoft applications, such as SQL Server are not supported on Unix platforms.
- Permissions are more complex on Unix, so make sure you budget time for permissions issues.
- Ensure that clients, such as JDBC, ODBC, or Oracle are properly installed and configured on the Unix system.
- You are working with 32-bit libraries, so ensure that you do not have 64-bit conflicts.
- Ensure that files on Unix do not use CR-LF when they should use LF in lookup, log, reject, and code module files. When available, set the system default to record-separated.

Design Tips

- Move files in binary mode to the Unix box. For details, search for the keywords “Moving Files from Windows to Unix” in the online help.
- Use macros to initially design your transformations and processes in Windows. Then create macros to change paths to move to testing and production environments onto a Unix box. For details, see “Using Macros During Deployment”.
- Create a .djar file on Windows to run on Unix. The .djar file is a package that contains transformation and process files that can be run on the Data Integrator engine or Integration Manager. For details, search for the keywords “Unix .djar Life Cycle and Methodology” in the online help or the [sample](#) “Running a .djar File”.

Employing Code Modules and User-Defined Functions for Reusability

When designing transformations and processes, you can incorporate reusability into your solutions by using code modules and user-defined functions. Code modules are text files that contain RIFL code, a proprietary scripting language used throughout the Data Integrator tool set. User-defined functions are saved as .rifl code within the code module.

To understand this topic, you should be familiar with the following:

- Data Integrator functionality
- Basic Rapid Integration Flow Language (RIFL) scripting
- Text file manipulation

In most instances, RIFL code is applied at the target field expression level in individual data fields. By using code modules, you can write a function one time and apply the function globally across multiple maps and processes.

External code modules can be moved to any other machine with the Map Designer or Integration Engine. This allows you to develop a user-defined library for sharing among different members of your team.

Review the following sections which begin with an overview and then continue with code module and script examples.

“Code Module Use Cases”

“Creating Code Modules”

“User-Defined Functions and RIFL Scripts”

“Creating User-Defined Functions”

“Using Global Variables”

Code Module Use Cases

Often integration projects are broken up into many smaller units of work that are then assigned and completed by different team members. When in the project design stage, assess what functions are common to the data being transformed, so you can write one common function and apply it across many scenarios.

Typically, one code module is modified and checked into version control. This module can then be checked out to the various team members, ensuring that they have the latest version of the user-defined functions for their maps and processes.

Some of the more common uses for implementing code modules and user-defined functions include the following:

- Custom logging functions
- Custom string formatting
- Calling external managed code

For example, user-defined function can be written to accept a logging severity parameter of high, medium, or low. When specific data is transformed in a map and meets one of the three criteria, you can write separate log files depending upon how the function is implemented.

Creating Code Modules

➤ To create a code module in Map Designer

- 1** In the Transformation and Map Properties window, select **Code Modules**.
- 2** Select the paper icon in the upper right hand corner of the window.

RIFL script editor opens, where you can write user-defined functions. The benefit of this approach is that you can create your functions in the RIFL editor window, that includes color coded keywords, bookmarks, and on-screen help references for built-in functions.
- 3** When you are finished composing the code, select **File > Save Script to File**. Using this method automatically references the code module in the transformation you are currently working on.

In addition to using Map Designer to create code modules, you can use a simple text editor, such as Notepad, and save the file with a .rifl extension. Files with the extension .bas, .asc, and .txt can also be read by the Data Integrator compiler, but .rifl is the preferred extension.

Naming Code Modules

As a best practice, rename .rifl files to describe its function. For instance, use descriptive names such as .incore.rifl, .xlate.rifl, or .lookup.rifl.

Once you use this naming convention, you can open the files in RIFL script editor to see which files have the extra extension names. This way, you can use the list as a reference to find the external functions you created.

User-Defined Functions and RIFL Scripts

RIFL scripts are the foundation of user-defined functions. Note that RIFL does not support the programming concept of data types. This means that all variables are considered variant and are not declared as a certain type, which simplifies scripting.

RIFL supports both functions and subroutines, which are user-defined functions that may or may not return a value. RIFL is supported on both Windows and Unix platforms.

Creating User-Defined Functions

Now that you have created a code module .rifl file, you can create user-defined functions to be contained within the code module. To compile properly, user-defined functions must begin with a function declaration statement and end with the End Function statement.

User-defined functions cannot have the same name as a function already declared by Data Integrator, such as the Format function. While this is the only naming caveat, it is best to name functions using the camel case style familiar to Visual Basic programmers, for example, udfMyFunctionName().

The number of parameters set in the function declaration must be the same as the number of parameters passed when calling the function. For Map Designer to return a value, the function name must be set equal to the return value.

User-Defined Function Example 1

This user-defined function accepts one parameter and does not return a value.

```
public function udfWarnLogger(msg1)
    dim msg2, msg
    msg = msg1
    msg2 = "Warning Logger: " & msg
```

```
    LogMessage("WARN", msg2)
end function
```

User-Defined Function Example 2

This user-defined function accepts no parameters, but returns a value.

```
Public Function CreateGuid()
Dim typeLib As Object
Dim guid
Set typeLib = CreateObject("Scriptlet.TypeLib")
guid = typeLib.Guid
Set typeLib = Nothing
`Sets the return value
CreateGuid = guid
End Function
```

Note that the incoming parameter is referenced again within the function. This is required if the incoming parameter is not set up as a public or private level variable in your transformation. When declaring variables in user-defined functions, the default behavior is local in scope. Data Integrator tools allow variables to be declared with Dim (local), private, and public (global) statements.

Using Global Variables

Declare any global variables at the beginning of the .rifl file so that at run time, the compiler recognizes the variables as global. Also, be aware that the use of Option Explicit statements defined in your transformation implies this requirement on any referenced external code modules. It is best practice to write external code modules as if Option Explicit statement were declared.

Related Topics

Search for the following keywords in the online help:

- “Employing User-Defined Functions”
- “Calling User-Defined Functions”
- “Option Explicit Statement”
- “Calling External Code Modules”

Troubleshooting Tips for Process Designer

When developing and testing processes in Process Designer, you may encounter errors or other issues. For help, see the following resolutions and best practice tips.

Issue

Process Designer encounters an error and stops responding. The process includes nested loops, a combination of two outside loops, with an inner loop inside the right outer loop. After both outer loops are executed, another execution through the inner loop results in the engine being unable to find more steps to execute, so the process stops.

Resolution

Avoid nested loops when designing your processes. Rearrange the right outer and inner loop to eliminate the inner loop.

Issue

You lose connection option information for a transformation in a Process Designer Transformation step.

If you edit transformations used in more than one process with incorrect location information, session information becomes out of sync with the map file in a process step. Once sessions are out of sync, you may lose connection information, such as IDs and passwords.

Resolution

To restore connection information, perform the following steps:

- 1** Open each transformation in Map Designer (not in Process Designer), and save each transformation with a unique name.
- 2** Make any necessary changes, ensure your changes are correct, and save each transformation.
- 3** Delete the previous transformation files (tf.xml and map.xml) with the old names.
- 4** Open the new transformations and save each transformation as the old names.

- 5 Remove the original Transformation steps from Process Designer, then add new transformation steps and reassociate the new transformations to each new process step.

Related Best Practice Tips

- After you move or copy transformation files, always reassociate the transformation to the related process steps.
- Be careful when editing maps that may be referenced in more than one process.
- Once a transformation has been added to Process Designer, open the transformation only from within Process Designer to edit it.

Issue

The Decision step always takes the same path.

Resolution

Your process cannot include duplicate step names, so ensure that all step names are unique. For additional file naming tips, see “Naming Conventions for Specification Files, Variables, and Objects”.

Issue

In a process that includes parallel Transformation steps, Process Designer stops responding.

Resolution

You must set up a separate session for each parallel process step. For details on running parallel maps, see “Using Process Designer to Run Maps on Multiple Threads” and “Managing SQL Sessions in Process Designer”.

Index

B

- best practices
 - dynamic sql lookups A-5

C

- comparing
 - multithreaded, multi-instance engine configurations 2-18

D

- dynamic SQL
 - comparing lookup methods A-5
 - incore lookup example 1-21

E

- engine
 - comparing configurations 2-18

F

- file naming conventions 1-4

I

- initial integration project checklists 1-2
- integration project scoping 1-2

L

- life cycles
 - development 1-11
 - production 1-15
 - test 1-14
- lookups A-5
 - choosing the best method A-5
 - comparing A-5
 - inline or search and replace A-5
 - methods, comparing A-5
 - SQL A-5
 - sql pass-through A-8
 - static flat file A-6

M

- macros
 - ChangeSource and ChangeTarget actions 1-19
 - command line 1-21
 - escape character for macros 1-21
 - for storing expressions 1-20
 - scenarios 1-22
 - specifying file paths 1-18
 - testing return values 1-20
 - using macroexpand function in incore lookups 1-21
- multiple threads
 - running maps in Process Designer 2-8

N

- naming conventions
 - files 1-4
 - macros 1-5
 - variables, functions 1-4

P

- performance
 - designing faster transformations, processes 2-5
 - enhancing run-time 2-2
- process
 - multi-instance definition 2-9
 - multithreaded definition 2-8
 - single threaded definition 2-8
- process designer
 - running maps on multiple threads 2-8

R

- repositories, specifying 1-9
- Repository Explorer
 - collections 1-9
 - defining workspaces 1-6
 - specifying repositories 1-9
- run-time performance
 - improving 2-2

S

SQL, lookups A-5

T

test data, generating A-13

transformation design improvements 2-5

troubleshooting

 reading the error log 3-15

U

Unix, Windows deployments, comparing A-14

W

workspaces, defining 1-6