

What is the MD5 Function in Informatica?

MD5 (Message Digest Function) is a hash function in Informatica which is used to evaluate data integrity.  The MD5 function uses Message-Digest Algorithm 5 (MD5) and calculates the checksum of the input value.  MD5 is a one-way cryptographic hash function with a 128-bit hash value.

MD5 returns a 32 character string of hexadecimal digits 0-9 & a-f and returns NULL if the input is a null value.

Example:

When you wish to write changed data to a database.  Use MD5 to generate checksum values for rows of data that you read from a source.

When you run a session, compare the previously generated checksum values against the new checksum values.  Then, write the rows with an updated checksum value to the target.  You can conclude that an updated checksum value would indicate that the data has changed.

Change data capture (CDC) can be done in many ways.  There are methodologies such as *Timestamp, Versioning, Status indicators, Triggers and Transaction* logs and *Checksum*.  The advantage of using MD5 function is to reduce overall extract-transform-load (ETL) run-time and the cache memory usage, by caching only the required fields which are of utmost importance.

When To Use MD5 Function in Informatica?

Only update the changed records (any column change) in the target.  Instead of passing all existing records to the target for update, ( whether changed or unchanged ) it's always recommended to compare the records.

Column-to-column comparison is little painful if your column counts are high.  Concatenate all columns and use the MD5 Function (source and target records for the same key) and then compare the output from the MD5 Function.  The changed records can be identified and only those records can be updated into the target.  This calculation is done really fast and the output of MD5 Function can be used as a unique key to differentiate records

MD5 will help in improving performance when compared to lookups only if the comparison columns are more than 10.  MD5 function enhances the performance as compared to lookups only when the comparison columns are more than 10.

Limitations

If we have non-string columns (SMALLINT, INT, NUMBER, etc.) we need to convert them into characters using TO\_CHAR function because the MD5 Function only validates character strings.  If you don't use the TO\_CHAR, the output port using MD5 may become invalid.  Sometimes it does validate the output port using MD5, but the warning message generated may read validated and non-string data type (e.g. SMALLINT) columns are converted to string.  This warning has a great performance impact.

Normal Approach

To identify records for updates and inserts, we use a lookup transformation.  The cache built by lookup depends on two factors:

1. The number of columns in the comparison condition
2. The amount of data in the lookup table.

When there is not a primary key column to identify the changes, there are two options:

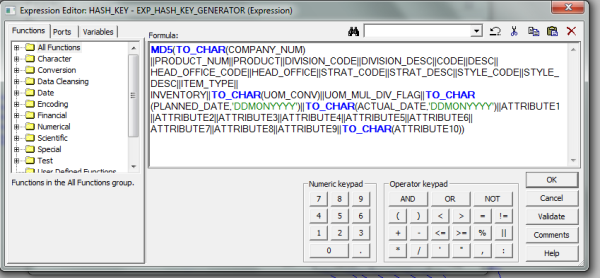
1. Compare all the columns in the lookup or...
2. Compare the data using the concept of power exchange change data capture.

Beware that this can degrade the performance.  In this scenario, using the MD5 Function is our best option.

Scenario

Consider a scenario where the incoming Product records consist of PRODUCT\_NUM, PRODUCT\_DESC, and address fields which have no primary key are associated with them.  In such a scenario, it is imperative that a unique identifier be assigned to these records on-the-fly which is immutable.  This on-the-fly unique identifier can also be used in future loads as a key to identify whether an incoming customer record is a potential update or an insert.

The MD5 Function generates a unique hexadecimal string 32 which is character-wide for a given input string.  In this example, the source table of a customer include customer details such as PRODUCT\_NUM, PRODUCT\_DESC, and address fields.  The MD5 Function in an expression is used to assign a 32 character-wide key to each of these records and load them in target file.



Conclusion

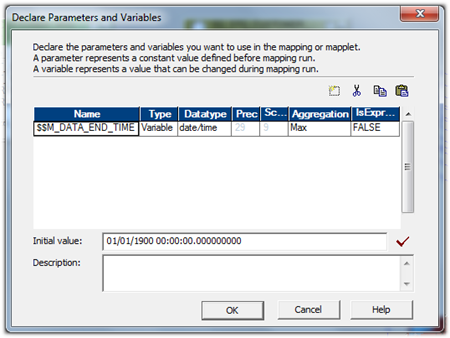
By using the MD5 values, we can identify whether the data is changed or unchanged without the performance being degraded and data is handled in the most efficient way possible. The MD5 value is always recommended for scenarios with many comparison columns and no primary key columns in the lookup table.   There is limitation, however; the input to the MD5 values needs to be a string by data type and it returns a 32 bit hexadecimal.

**Change Data Capture Implementation**

Now we understand the mapping variables, lets go ahead and start building our mapping with [Change Data Capture](http://www.disoln.org/2013/03/Change-Data-Capture-Implementation-for-Multi-Sourced-ETL-Process.html). Here we are going to implement [Change Data Capture](http://www.disoln.org/2012/12/Change-Data-Capture-CDC-Implementation-Using-CHECKSUM-Number.html) for CUSTOMER data load. We need to load any new customer or changed customers data to a flat file. Since the column UPDATE\_TS value changes for any new or updated customer record, we will be able to find the new or changed customer records using UPDATE\_TScolumn. 

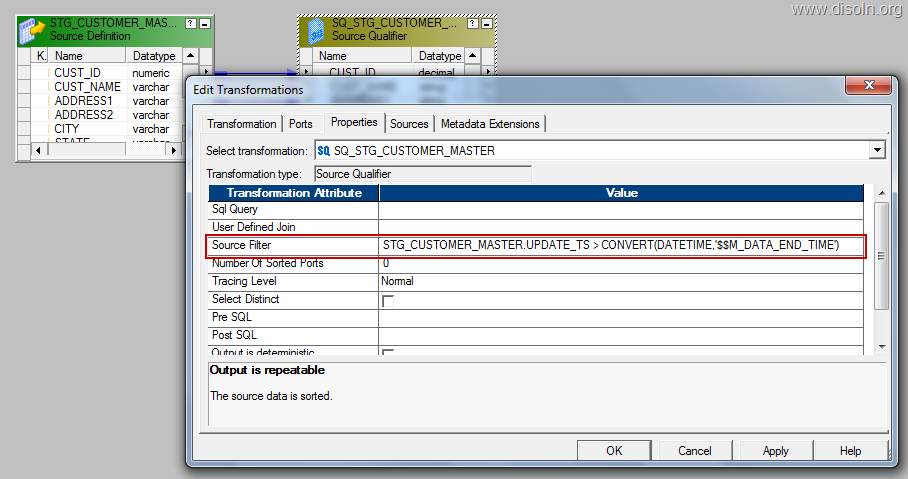
As the first step lets start the mapping and create a mapping variable as shown in below image.

* + $$M\_DATA\_END\_TIME as Date/Time

[](http://lh6.ggpht.com/-_sHvDtOQOIM/UHes3ESSEjI/AAAAAAAAFp8/cA8W_xx8ELE/s1600-h/image%25255B11%25255D.png)

Now bring in the source and source qualified to the mapping designer workspace. Open the source qualifier and give the filter condition to get the latest data from the source as shown below.

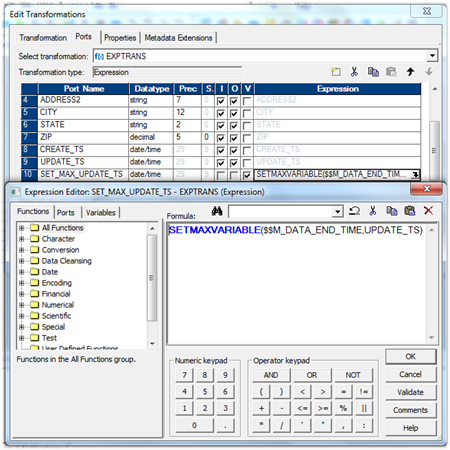
* + STG\_CUSTOMER\_MASTER.UPDATE\_TS > CONVERT(DATETIME,'$$M\_DATA\_END\_TIME')



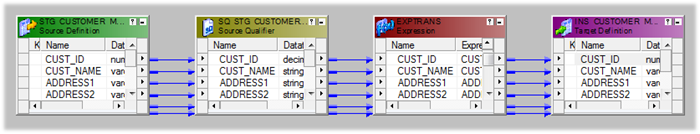
Note : This filter condition will make sure that, latest data is pulled from the source table each and every time. Latest value for the variable $M\_DATA\_END\_TIME is retrieved from the repository every time the session is run. 

Now map the column UPDATE\_TS to an expression transformation and create a variable expression as below.

* + SETMAXVARIABLE($M\_DATA\_END\_TIME,UPDATE\_TS)

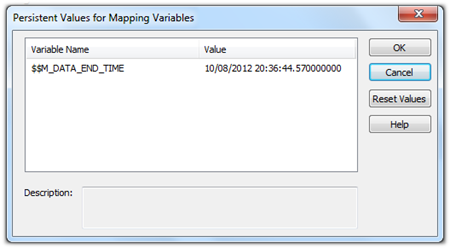
[](http://lh5.ggpht.com/-wkXWeSOXjU0/UHes30RqY7I/AAAAAAAAFqM/tK8ZTh6CFnU/s1600-h/image%25255B20%25255D.png)

Note : This expression will make sure that, latest value from the the column UPDATE\_TS is stored into the repository after the successful completion of the session run.Now you can map all the remaining columns to the down stream transformation and complete all other transformation required in the mapping.

[](http://lh3.ggpht.com/-bqC0GTJxHxU/UHes4_NIJfI/AAAAAAAAFqc/IHzZOZq_gcY/s1600-h/image%25255B25%25255D.png)

That’s all you need to configure Change Data Capture, Now create your workflow and run the workflow. Once you look into the session log file you can see the mapping variable value is retrieved from the repository and used in the source SQL, just like shown in the image below.[http://lh6.ggpht.com/-boEI6Hod5wo/UHes6ZXxL9I/AAAAAAAAFq0/vFnBgzKrMmc/image_thumb%25255B15%25255D.png?imgmax=800](http://lh4.ggpht.com/-f79dyLcritU/UHes5y55zII/AAAAAAAAFqs/lKfOaVYlUfs/s1600-h/image%25255B29%25255D.png)

You can look at the mapping variable value stored in the repository, from workflow manager. Choose the session from the workspace, right click and select 'View Persistent Value'. You get the mapping variable in a pop up window, like shown below.

[](http://lh5.ggpht.com/--J6ZUQepFhM/UHes6txLX7I/AAAAAAAAFq8/cTRXl0Fs7Wk/s1600-h/image%25255B7%25255D.png)

**Data Masking in Informatica**

**Data Masking** is one of the latest features introduced  by [Informatica](http://dwhlaureate.blogspot.in/2012/08/informatica-9-features.html" \t "_blank)and is big advantage considering the security of sensitive data stored in [Datawarehouse](http://www.blogger.com/%20http:/dwhlaureate.blogspot.in/2012/07/datawarehouse.html" \t "_blank).For eg

* SSN number of a customer
* Credit Card number
* Phone number etc…

This is done by masking the original data using various masking techniques   i.e. converting the original data into some other readable format. This is done without affecting the granularity of the table ,all the integrity constraints are kept the same.The objective behind using data masking transformation is to restrict the outside world from accessing the Customer Information.

Since data ,masking is an integrated option available in **@Informatica Power centre** it greatly reduces the Development effort.

**Why we need data masking?**

Normally in projects we have non production environments like development, testing and training . Often the IT companies are careless about the security  in the developments phase. While moving the code into actual production environment they rarely notice that the information can be leaked out.So we need to be careful here.In order to avoid this situation we can do masking at the development phase itself and then outsource the data.That we will be more safer.

Another advantage is that we can avoid the use of Dummy values in the testing and development.Instead of using Dummy values we can use the actual and then mask it and the successfully promote to the Live Environment.

Now we will go through the different algorithms used for data masking

* **Nondeterministic randomization**:Replace a sensitive field with a randomly generated .This can be some either any number or date field.
* **Blurring:** Add a random variance to the original value .Variance can be small or larger. For example adding 10 percent increase in the salary of employee.
* **Repeatable masking:** Maintain referential integrity by using values that are unique and different.Eg Replace the SSN number 1/45/89 with 12/22/44
* **Substitution:**  Randomly substitute original values with entirely different but real valuesE.g., substitute “Wayne Smith” with “Don Kelly”)

[Data quality](http://www.disoln.org/2012/10/User-Defined-Error-Handling-in-Informatica-PowerCenter.html) is very critical to the success of every data warehouse projects. So ETL Architects and Data Architects spent a lot of time defining the [error handling](http://www.disoln.org/2012/07/error-handling-made-easy-using.html)approach. Informatica PowerCenter is given with a set of options to take care of the error handling in your ETL Jobs. In this article, lets see how do we leverage the PowerCenter options to handle your exceptions.

**Error Classification**

You have to deal with different type of errors in the ETL Job. When you run a session, the PowerCenter Integration Service can encounter fatal or non-fatal errors. Typical error handling includes:

* + **User Defined Exceptions**: Data issues critical to the data quality, which might get loaded to the database unless explicitly checked for quality. For example, a credit card transaction with a future transaction data can get loaded into the database unless the transaction date of every record is checked.
  + **Non-Fatal Exceptions**: Error which would get ignored by Informatica PowerCenter and cause the records dropout from target table otherwise handled in the ETL logic. For example,  a data conversion transformation error out and fail the record from loading to the target table.
  + **Fatal Exceptions**: Errors such as database connection errors, which forces Informatica PowerCenter to stop running the workflow.

**I. User Defined Exceptions**



Business users define the user defined user defined exception, which is critical to the data quality. We can setup the user defined error handling using;

* + - 1. [Error Handling Functions](http://www.disoln.org/2012/10/User-Defined-Error-Handling-in-Informatica-PowerCenter.html).
      2. User Defined Error Tables.

**1. Error Handling Functions**

We can use two functions provided by Informatica PowerCenter to define our user defined error capture logic.

**ERROR()** : This function Causes the PowerCenter Integration Service to skip a row and issue an error message, which you define. The error message displays in the session log or written to the error log tables based on the error logging type configuration in the session.

You can use ERROR in Expression transformations to validate data. Generally, you use ERROR within an IIF or DECODE function to set rules for skipping rows.

Eg : IIF(TRANS\_DATA > SYSDATE,ERROR('Invalid Transaction Date'))

Above expression raises an error and drops any record whose transaction data is greater than the current date from the ETL process and the target table.

**ABORT()**: Stops the session, and issues a specified error message to the session log file or written to the error log tables based on the error logging type configuration in the session. When the PowerCenter Integration Service encounters an ABORT function, it stops transforming data at that row. It processes any rows read before the session aborts.

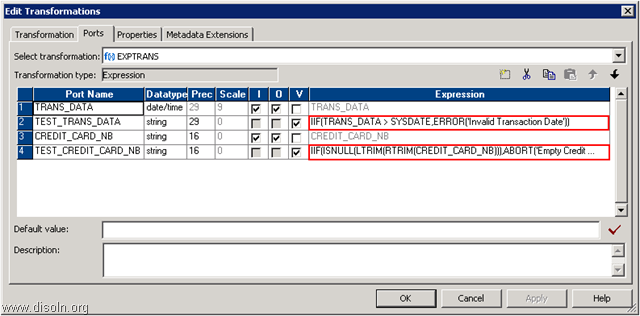
You can use ABORT in Expression transformations to validate data.

Eg : IIF(ISNULL(LTRIM(RTRIM(CREDIT\_CARD\_NB))),ABORT('Empty Credit Card Number'))

Above expression aborts the session if any one of the transaction records are coming with out a credit card number.

**Error Handling Function Use Case**

Below shown is the configuration required in the expression transformation using ABORT() and ERROR() Function. This transformation is using the expressions as shown in above examples.

[](http://lh4.ggpht.com/-Ps9JKWcCkZ8/U0JGFXzYFyI/AAAAAAAAJS4/y5P3ShnPgoU/s1600-h/image%25255B36%25255D.png)

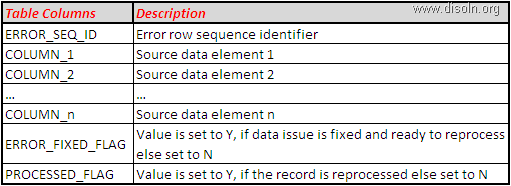
**Note**:- You need to use these two functions in a mapping along with a session configuration for [row error logging](http://www.disoln.org/2012/07/error-handling-made-easy-using.html) to capture the error data from the source system. Depending on the session configuration, source data will be collected into Informatica predefined PMERR error tables or files.

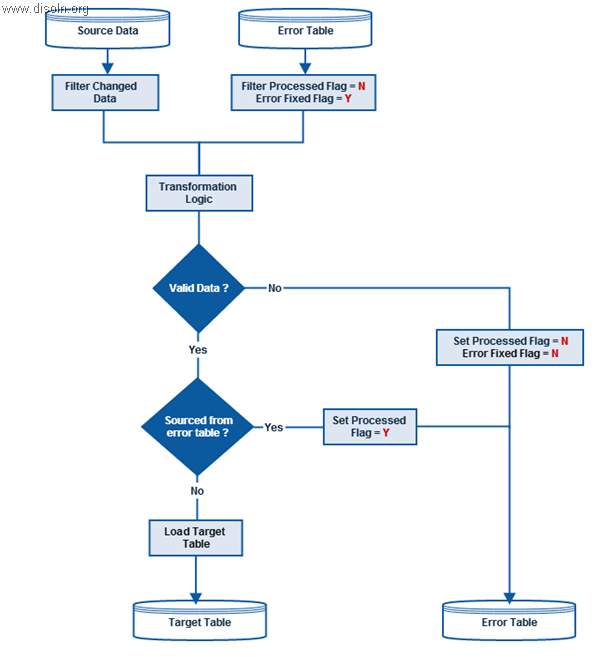
Please refer the article "[User Defined Error Handling in Informatica PowerCenter](http://www.disoln.org/2012/10/User-Defined-Error-Handling-in-Informatica-PowerCenter.html)**"**for more detailed level implementation information on user defined error handling.

**2. User Defined Error Tables**

Error Handling Functions are easy to implement with very less coding efforts, but at the same time there are some disadvantages such as readability of the error records from the PMERR tables and [performance](http://www.disoln.org/search/label/Performance%20Tips?max-results=8) impact. To avoid the disadvantages of error handling functions, you can create your own error log tables and capture the error records into it.

Typical approach is to create an error table which is similar in structure to the source table. Error tables will include additional columns to tag the records as "error fixed", "processed". Below is a sample error table. This error table includes all the columns from the source table and additional columns to identify the status of the error record.

[](http://lh3.ggpht.com/-SXg7jFaTW9I/U0Ixcb-DGxI/AAAAAAAAJR8/LOamayJqQ0E/s1600-h/image%25255B12%25255D.png)

Below is the high level design.[](http://lh5.ggpht.com/-eVoD2A6KJec/U0JCLWmtkAI/AAAAAAAAJSU/XqiOP7reDZM/s1600-h/Error%252520Processing%25255B12%25255D.png)

Typical [ETL Design](http://www.disoln.org/search/label/ETL%20Design?&max-results=15) will read error data from the error table along with the source data. During the data transformation, data quality will be checked and any record violating the quality check will be moved to error tables. Record flags will be used to identify the reprocessed and records which are fixed for reprocessing.

**II. Non-Fatal Exceptions**



Non-fatal exception causes the records to be dropped out in the ETL process, which is critical to quality. You can handle non-fatal exceptions using;

* + - 1. Default Port Value Setting.
      2. Row Error Logging.
      3. Error Handling Settings.

**1. Default Port Value Setting**

Using default value property is a good way to handle exceptions due to NULL values and unexpected transformation errors. The Designer assigns default values to handle null values and output transformation errors. [PowerCenter Designer](http://www.disoln.org/2012/08/Understand-Informatica-PowerCenter-Mapping-Designer.html) let you override the default value in input, output and input/output ports.

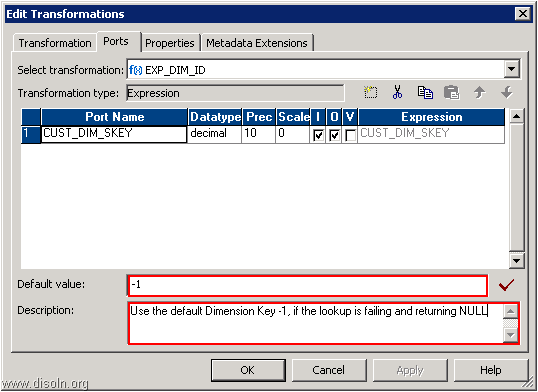
Default value property behaves differently for different port types;

* **Input ports** : Use default values if you do not want the Integration Service to treat null values as NULL.
* **Output ports** : Use default values if you do not want to skip the row due to transformation error or if you want to write a specific message with the skipped row to the session log.
* **Input/output ports** : Use default values if you do not want the Integration Service to treat null values as NULL. But no user-defined default values for output transformation errors in an input/output port.

**Default Value Use Case**

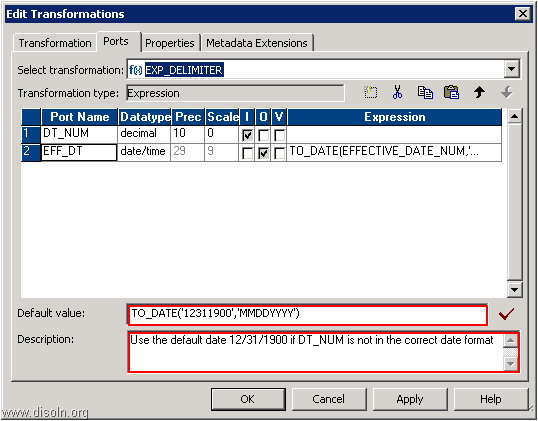
**Use Case 1**

Below shown is the setting required to handle NULL values. This setting converts any NULL value returned by the [dimension](http://www.disoln.org/2012/08/slowly-changing-dimension-type-2-implementation-using-informatica.html) lookup to the default value -1. This technique can be used to handle [late arriving dimensions](http://www.disoln.org/2013/12/Design-Approach-to-Handle-Late-Arriving-Dimensions-and-Late-Arriving-Facts.html)

[](http://lh3.ggpht.com/-iAbkIOrccS8/U0JCMQsTnNI/AAAAAAAAJSg/t8MWICQXrKs/s1600-h/image%25255B25%25255D.png)

**Use Case 2**

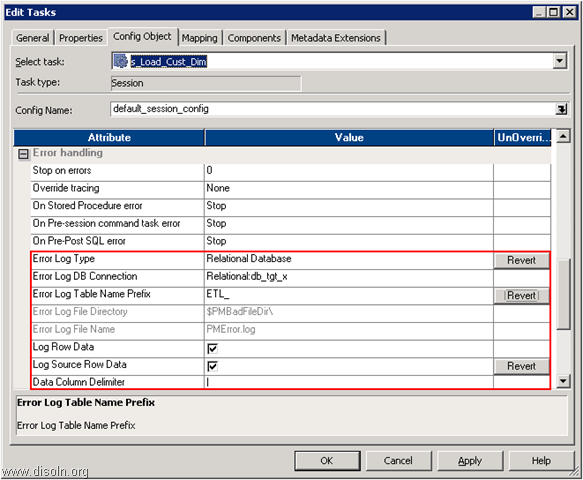
Below setting uses the default expression to convert the date if the incoming value is not in a valid date format.

[](http://lh4.ggpht.com/-Py56aSkSqLg/U0JSu_emPiI/AAAAAAAAJTQ/L-y8fPez8Vo/s1600-h/image%25255B47%25255D.png)

**2. Row Error Logging**

Row error logging helps in capturing any exception, which is not consider during the design and coded in the mapping. It is the perfect way of capturing any unexpected errors.

Below shown session error handling setting will capture any un handled error into PMERR tables.

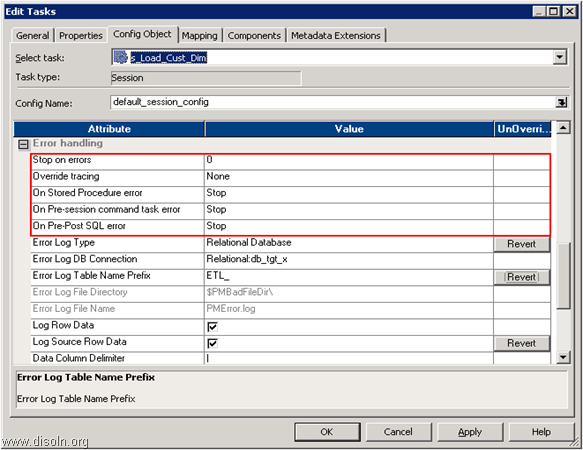
[](http://lh5.ggpht.com/-faZ4fR4Q-X8/U0JVzfnxdtI/AAAAAAAAJTk/txvtOHH0LQY/s1600-h/image%25255B62%25255D.png)

Please refer the article [Error Handling Made Easy Using Informatica Row Error Logging](http://www.disoln.org/2012/07/error-handling-made-easy-using.html) for more details.

**3. Error Handling Settings**

Error handling properties at the session level is given with options such as Stop On Errors, Stored Procedure Error,  Pre-Session Command Task Error and Pre-Post SQL Error. You can use these properties to ignore or set the session to fail if any such error occurs.

* **Stop On Errors** : Indicates how many non-fatal errors the Integration Service can encounter before it stops the session.
* **On Stored Procedure Error** : If you select Stop Session, the Integration Service stops the session on errors executing a pre-session or post-session stored procedure.
* **On Pre-Session Command Task Error** : If you select Stop Session, the Integration Service stops the session on errors executing pre-session shell commands.
* **Pre-Post SQL Error** : If you select Stop Session, the Integration Service stops the session errors executing pre-session or post-session SQL.

[](http://lh3.ggpht.com/-qwTYuhVWnmA/U0JV0fNHNaI/AAAAAAAAJT0/YENBCiMO7dc/s1600-h/image%25255B73%25255D.png)

**III. Fatal Exceptions**

A fatal error occurs when the Integration Service cannot access the source, target, or repository. When the session encounters fatal error, the PowerCenter Integration Service terminates the session. To handle fatal errors, you can either use a [restartable ETL design](http://www.disoln.org/2013/02/Restartability-Design-for-Different-Type-ETL-Loads.html) for your workflow or use the [workflow recovery features](http://www.disoln.org/2013/07/Workflow-Recovery-Configuration-for-Informatica-PowerCenter-Workflows.html) of Informatica PowerCenter

* + - 1. [Restartable ETL Design](http://www.disoln.org/2013/02/Restartability-Design-for-Different-Type-ETL-Loads.html)
      2. [Workflow Recovery](http://www.disoln.org/2013/07/Workflow-Recovery-Configuration-for-Informatica-PowerCenter-Workflows.html)

**1. Restartable ETL Design**

Restartability is the ability to restart an ETL job if a processing step fails to execute properly. This will avoid the need of any manual cleaning up before a failed job can restart. You want the ability to restart processing at the step where it failed as well as the ability to restart the entire ETL session.

Please refer the article "[Restartability Design Pattern for Different Type ETL Loads](http://www.disoln.org/2013/02/Restartability-Design-for-Different-Type-ETL-Loads.html)**"**for more details on restartable ETL design.

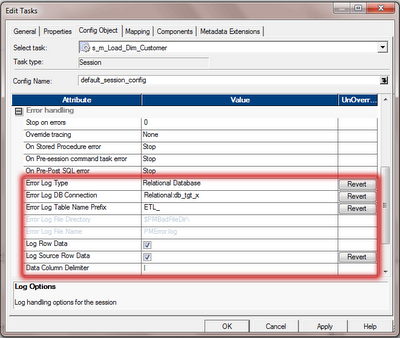
**2. Workflow Recovery**

Workflow recovery allows you to continue processing the workflow and workflow tasks from the point of interruption. During the workflow recovery process Integration Service access the workflow state, which is stored in memory or on disk based on the recovery configuration. The workflow state of operation includes the status of tasks in the workflow and workflow variable values.

Every time we start with a Data Warehouse or Data Integration project we spent lot of time defining our error handling approach. Still there are good chances that we might miss some scenarios because of unexpected data issues. Here in this article, lets us discuss an approach to handle the unexpected error.

Error handling Configuration.

We do not have any error configuration required during the mapping development. So after the mapping is created, during the session configuration set the session properties as shown in below image.

[](http://3.bp.blogspot.com/-EBtdaXQ-iNs/UANftysNu5I/AAAAAAAAEd4/bk_M9YC_940/s1600/error-handling-config.png)

Here is the details on what each property means.

Error Log Type :- Specifies the type of error log to create. It can be relational database or flat file.

Error Log DB Connection :- Database connection for a relational log.

Error Log Table Name Prefix :- Specifies the table name prefix for relational logs.

Log Row Data :- Specifies whether or not to log transformation row data.

Log Source Row Data :- Specifies whether or not to log source row data.

Data Column Delimiter :- Data will be delimited by the specified character in DB column.

With this configuration we specified, Informatica PowerCenter will create four different tables for error logging and the table details as below.

ETL\_PMERR\_DATA :- Stores data about a transformation row error and its corresponding source row.

ETL\_PMERR\_MSG :- Stores metadata about an error and the error message.

ETL\_PMERR\_SESS :- Stores metadata about the session.

ETL\_PMERR\_TRANS :- Stores metadata about the source and transformation ports, when error occurs.

With this configuration, we are done with the setting required to capture any error during the session execution. Now lets see how do we retrieve the data from the error log tables and report it to Business Users or IT Department.

Report the Error Data.

Now we have the error data stored in the error table, it is important to share the error data to the Business Users or to IT Department. Lets see how we can pull data from these tables.We can pull the basic error report using the SQL. We can be more fancy with the SQL and get more information from the error tables.

select

sess.FOLDER\_NAME as 'Folder Name',

sess.WORKFLOW\_NAME as 'WorkFlow Name',

sess.TASK\_INST\_PATH as 'Session Name',

data.SOURCE\_ROW\_DATA as 'Source Data',

msg.ERROR\_MSG as 'Error MSG'

from

ETL\_PMERR\_SESS sess

left outer join ETL\_PMERR\_DATA data

on data.WORKFLOW\_RUN\_ID = sess.WORKFLOW\_RUN\_ID and

data.SESS\_INST\_ID = sess.SESS\_INST\_ID

left outer join ETL\_PMERR\_MSG msg

on msg.WORKFLOW\_RUN\_ID = sess.WORKFLOW\_RUN\_ID and

msg.SESS\_INST\_ID = sess.SESS\_INST\_ID

where

sess.FOLDER\_NAME = <Project Folder Name> and

sess.WORKFLOW\_NAME = <Workflow Name> and

sess.TASK\_INST\_PATH = <Session Name> and

sess.SESS\_START\_TIME = <Session Run Time>

The above SQL can be converted to an Informatica mapping and can be scheduled to run after all your ETL jobs are completed to get a daily error report and the same can be scheduled for an email delivery to the data owners to take relevant action on the errors.

Pros and Cons of this Approach.

We should know the Pros and Cons of this approach before applying this to your project.

Pros.

Out of the box Solution Provided by Informatica.

Captures all the Error, Including the unthought error.

Less Coding and Testing efforts required by the development team.

Cons.

If an error occurs after an active transformation, Integration service will not be able to capture source data into the erro table.

In sessions with multiple non-pass through partitions, Integration service can not capture source data into error table.

Added overhead to the Session performance (Which is expected and acceptable).

Please leave us a comment below, if you have any difficulties implementing this error handling approach. We will be more than happy to help you.