# The SeETL RunTime

# User Guide

Version 3.2.00
1st January 2019
Instant Business Intelligence
info@instantBI.com

# Table of Contents


1. CHANGE CONTROL LOG ..... 6

2. AUDIENCE ..... 7

3. INTRODUCTION ..... 8

4. WHAT SEETL RUNTIME IS ..... 9

5. WHAT SEETL RUNTIME IS NOT ..... 9

6. STAR SCHEMA MANAGEMENT ..... 10
   6.1. Input Record and Fact Tables ..... 10
   6.2. Dimension Tables ..... 11
      6.2.1. Summary Levels in Dimension Tables ..... 11

7. COMPONENTS OF SEETL RUNTIME ..... 15
   7.1. Program Processes ..... 15
   7.2. Tables Required in the Target Database ..... 20
   7.3. Message Tables ..... 20
      7.3.1. ctl_audit_table ..... 20
      7.3.2. ctl_message_table ..... 21
   7.4. Control Tables ..... 22
      7.4.1. ctl_last_key_used ..... 22
      7.4.2. ctl_aggregation_control ..... 23
      7.4.3. ctl_dim_table_key_definitions ..... 25
      7.4.4. ctl_dim_table_type2_col_defs ..... 27
      7.4.5. ctl_dim_table_load_control ..... 28
         *7.4.5.1. Guidelines on In Memory Array Allocations ..... 31*
      7.4.6. ctl_batch_control ..... 34
      7.4.7. ctl_month_control ..... 36
      7.4.8. ctl_codes_lookup ..... 37
      7.4.9. ctl_mapping_ss ..... 38
      7.4.10. ctl_file_cycle_control ..... 39
      7.4.11. ctl_sequence_nums ..... 40
   7.5. DDL For Views ..... 41
      7.5.1. Fact Table Processing ..... 41
         *7.5.1.1. Input Table/View Naming Conventions ..... 41*
         *7.5.1.2. Dimension View Lookup Naming Conventions ..... 42*
         *7.5.1.3. Detail Fact Table Naming Conventions ..... 43*
         *7.5.1.4. Summary Fact Table Naming Conventions ..... 44*
         *7.5.1.5. Sort Work Tables Naming Conventions ..... 45*
      7.5.2. Dimension Table Processing ..... 46
         *7.5.2.1. Type 1 Dimension Processing Naming Conventions ..... 46*
         *7.5.2.2. Type 2 Dimension Processing Naming Conventions ..... 48*
      7.5.3. Association Table Processing ..... 51
         *7.5.3.1. Introduction to Associations ..... 51*
         *7.5.3.2. Association Tables – A Better Way ..... 52*
         *7.5.3.3. What is an Association Table? ..... 53*
         *7.5.3.4. Association Table Naming Conventions ..... 53*

*7.5.3.5. Association Table Input Table Naming Conventions ..... 54*
*7.5.3.6. Association Tables View Naming Conventions ..... 54*
*7.5.3.7. Other Interesting Aspects of Association Tables ..... 58*
*7.5.3.8. Running an Association Table Processing Program ..... 59*
*7.5.3.9. Example Data From An Association Table ..... 60*

8. PARAMETERS ..... 61
8.1. Definition of Parameters ..... 61
8.2. Parameters Specific to Individual Programs ..... 80
8.3. Parameters Specific to Memory Mapped IO Processing ..... 84

9. ISSUING COMMANDS TO RUN SEETL ..... 85
9.1. CTLDM01 – Type 1 Dimension Maintenance ..... 85
9.2. CTLDM02 – Type 2 Dimension Maintenance ..... 86
9.3. CTLAT01 – Attribute a Detailed Input Table ..... 87
9.4. CTLAG01 – Aggregate an Attributed Fact Working File ..... 88
9.5. CTLCL01 – Consolidate Aggregated File with the Summary Fact Table ..... 89
9.6. Loading Data into the Data Warehouse ..... 90

10. INSTALLING SEETL RUNTIME ..... 92
10.1. Step 1 – Unzip the zip file and Install ..... 92
10.2. Step 2 – Restore the SQL Server 2005 database SEETL3000 ..... 93
10.3. Step 3 – Give Userid dba DBA access to SEETL3000 ..... 93
10.4. Step 4 – Create ODBC Data Source to SEETL3000 ..... 93

11. LIMITATIONS OR CONSTRAINTS ..... 94
11.1. General Limitations or Constraints ..... 94
11.2. Operating Sytems and Database Supported ..... 96
11.3. MetaData Reports ..... 98
11.4. Other Limitations ..... 99

12. DATA WAREHOUSING UTILITIES ..... 100
12.1. Why Have Data Warehousing Utilities? ..... 100
12.2. What Utilities are Available? ..... 101
12.3. What Components are Common across the SeETL[RT]? ..... 102
12.4. Tables Required in Target Database ..... 103
12.5. The Data Transfer Utility ..... 104
12.5.1. Why Have the Data Transfer Utility? ..... 104
12.5.2. What Does the Data Transfer Utility do? ..... 105
12.5.3. How Does the Data Transfer Utility Determine Primary Keys? ..... 106
12.5.4. Tables Required in Target Database ..... 106
12.5.5. Why Have a Self Describing File Format for the Data Transfer Utility? ..... 107
12.5.6. What Does the Self Describing File Format Look Like? ..... 108
12.5.7. Examples of invoking the Data Transfer Utility ..... 109
12.6. Batch Maintenance Utility ..... 111
12.6.1. Why Have the Batch Maintenance Utility? ..... 111
12.6.2. What Does the Batch Maintenance Utility Do? ..... 111
12.6.3. Tables Required in the Target Database ..... 112
12.6.4. Examples of Invoking the Batch Maintenance Utility ..... 112
12.7. SQL Statement Processor Utility ..... 113
12.7.1. Why Have the SQL Statement Processor Utility? ..... 113
12.7.2. What Does the SQL Statement Processor Utility Do? ..... 113
12.7.3. Tables Required in the Target Database ..... 113
12.7.4. Examples of Invoking the SQL Statement Processor Utility ..... 113

12.8. The SQL Server 2000 DDL Generator Utility ..... 114
12.8.1. Why Have The SQL Server 2000 DDL Generator Utility? ..... 114
12.8.2. What Does The SQL Server 2000 DDL Generator Utility Do? ..... 114
12.8.3. Tables Required in the Target Database ..... 114
12.8.4. Example of Invoking The SQL Server 2000 DDL Generator Utility ..... 117
12.9. The Delimiter Separated Values Reformat Utility ..... 118
12.9.1. Why Have the Delimiter Separated Values Reformat Utility? ..... 118
12.9.2. What Does the Delimiter Separated Values Reformat Utility do? ..... 118
12.9.3. Tables Required in Target Database ..... 118
12.9.4. Examples of invoking the Delimiter Separated Values Reformat Utility ..... 119
12.10. The Fixed File Format Reformat Utility ..... 120
12.10.1. Why Have the Fixed File Format Reformat Utility? ..... 120
12.10.2. What Does the Fixed File Format Reformat Utility do? ..... 120
12.10.3. Tables Required in Target Database ..... 120
12.10.4. Examples of invoking the Fixed File Format Reformat Utility ..... 121
12.11. The Generate Delta File Utility ..... 123
12.11.1. Why Have the Generate Delta File Utility? ..... 123
12.11.2. What Does the Generate Delta File Utility do? ..... 124
12.11.3. Tables Required in Target Database ..... 129
12.11.4. Examples of invoking the Generate Delta File Utility ..... 130
12.12. Batch Processing Scheduling Utility ..... 138
12.12.1. Why Have the Batch Processing Scheduling Utility? ..... 138
12.12.2. Tables Required in Target Database ..... 139
*12.12.2.1. The Batch Control Table ..... 139*
*12.12.2.2. The Batch Pre-Requisite Table. ..... 140*
*12.12.2.3. The Process Group Pre-Requisites Table ..... 143*
*12.12.2.4. The Commands Table ..... 145*
*12.12.2.5. The Process Commands Table ..... 147*
*12.12.2.6. The Batch Run Log Table ..... 149*
*12.12.2.7. The Process Group Run Log Table ..... 150*
*12.12.2.8. The Process Run Log Table ..... 151*
12.12.3. What Does the Batch Processing Scheduling Utility do? ..... 152
12.12.4. How Does the Batch Processing Scheduling Utility Work? ..... 156
12.12.5. Examples of Invoking the Batch Processing Scheduling Utility ..... 157
12.12.6. What is the Process Group Manager (CTLU009) ..... 158
12.13. DataStage Job Submission Utility – No Parameters ..... 161
12.13.1. Why Have the DataStage Job Submission Utility? ..... 161
12.13.2. What Does the DataStage Job Submission Utility Do? ..... 161
12.13.3. How Does the DataStage Job Submission Utility Work? ..... 162
12.13.4. Integration with the Batch Processing Scheduling Utility ..... 164
12.13.5. Tables Required in the Target Database ..... 165
12.13.6. Examples of Invoking the DataStage Job Submission Utility ..... 165
12.14. DataStage Job Submission Utility – With Parameters ..... 166
12.14.1. Why Have the DataStage Job Submission Utility – With Parameters? ..... 166
12.14.2. What Does the DataStage Job Submission Utility Do? ..... 166
12.14.3. How Does the DataStage Job Submission Utility Work? ..... 166
12.14.4. Tables Required in the Target Database ..... 167
12.14.5. Examples of Invoking the DataStage Job Submission Utility ..... 167
12.15. Load Dimension Tables into Memory Maps Utility ..... 168
12.15.1. Why Have the Load Dimension Tables into Memory Maps Utility? ..... 168
12.15.2. How Does the Load Dimension Tables into Memory Maps Utility Work? ..... 168
12.15.3. Tables Required in the Target Database ..... 169
12.15.4. Environment Variables Required in the Operating System ..... 169
12.15.5. Examples of Invoking the Load Dimension Tables into Memory Maps Utility ..... 169
12.16. MetaData Checking Utility ..... 171
12.16.1. Why Have the MetaData Checking Utility? ..... 171
12.16.2. How Does the MetaData Checking Utility Work? ..... 171
12.16.3. Tables Required in the Target Database ..... 171
12.16.4. Examples of Invoking the MetaData Checking Utility ..... 172
12.17. MetaData Printing Utility ..... 173
12.17.1. What Does the Metadata Printing Utility Do? ..... 173

12.17.2. How Does the MetaData Printing Utility Work? ....173
12.17.3. Tables Required in the Target Database ....173
12.17.4. Examples of Invoking the MetaData Printing Utility ....173
12.18. MetaData Loading Utility ....174
12.18.1. Why Have the MetaData Loading Utility? ....174
12.18.2. What Does the MetaData Loading Utility Do? ....174
12.18.3. How Does the MetaData Loading Utility Work? ....174
12.18.4. Tables Required in the Target Database ....175
12.18.5. Examples of Invoking the MetaData Loading Utility ....175
12.19. Data Correction Utility ....176
12.19.1. Why Have the Data Correction Utility? ....176
12.19.2. What Does the Data Correction Utility Do? ....176
12.19.3. How Does the Data Correction Utility Work? ....176
12.19.4. Data Correction Utility Parameters ....177
12.19.5. Tables Required in the Target Database ....182
12.19.6. Examples of Invoking the Data Correction Utility ....182
12.20. Batch Processing Sleep Utility ....183
12.20.1. Why Have the Batch Processing Sleep Utility? ....183
12.20.2. What Does the Batch Processing Sleep Utility Do? ....183
12.20.3. How Does the Batch Processing Sleep Utility Work? ....183
12.20.4. Tables Required in the Target Database ....183
12.20.5. Examples of Invoking the Batch Processing Sleep Utility ....183
12.21. XML File Reformat Utility ....184
12.21.1. Why Have the XML File Reformat Utility? ....184
12.21.2. What Does the XML File Reformat Utility Do? ....184
12.21.3. How Does the XML File Reformat Utility Work? ....185
12.21.4. Tables Required in the Target Database ....185
12.21.5. Examples of Invoking the XML File Reformat Utility ....185
12.22. Delete Target Rows Utility ....186
12.22.1. Why Have the Delete Target Rows Utility? ....186
12.22.2. What Does the Delete Target Rows Utility Do? ....186
12.22.3. How Does the Delete Target Rows Utility Work? ....186
12.22.4. Tables Required in the Target Database ....187
12.22.5. Examples of Invoking the Delete Target Rows Utility ....187
12.23. Bulk Load Rows into SQL Server Utility ....188
12.23.1. Why Have the Bulk Load Rows into SQL Server Utility? ....188
12.23.2. What Does the Bulk Load Rows into SQL Server Utility Do? ....188
12.23.3. How Does the Bulk Load Rows into SQL Server Utility Work? ....188
12.23.4. Tables Required in the Target Database ....189
12.23.5. Examples of Invoking the Bulk Load Rows into SQL Server Utility ....189
12.23.6. Bulk Load Rows into SQL Server Utility Parameters ....189
12.24. Execute ProcessName From Scheduler Utility ....191
12.24.1. Why Have the Execute ProcessName from Scheduler Utility? ....191
12.24.2. What Does the Execute ProcessName from Scheduler Utility Do? ....191
12.24.3. How Does the Execute ProcessName from Scheduler Utility Work? ....191
12.24.4. Tables Required in the Target Database ....191
12.24.5. Examples of Invoking the Execute ProcessName from Scheduler Utility ....192
12.24.6. Execute ProcessName from Scheduler Utility Parameters ....192
12.25. Batch Processing Scheduling Utility – Resilient Version ....194
12.25.1. Why Have the Batch Processing Scheduling Utility – Resilient Version? ....194
12.26. Execute SQL Statement and Logging Utility ....195
12.26.1. Why Have the Execute SQL Statement and Logging Utility? ....195
12.26.2. What Does the Execute SQL Statement and Logging Utility Do? ....195
12.26.3. Tables Required in the Target Database ....195
12.26.4. Examples of Invoking the Execute SQL Statement and Logging Utility ....198

# 1. CHANGE CONTROL LOG

| # | Date | Name | Description |
|---|---|---|---|
| 1.0 | 12/1/2003 | IBI Dev | Initial version for publication with SeETL[RT]. |
| 1.1 | 12/1/2003 | IBI Dev | Documentation review and additional information based on testing of Oracle8 as a source and target for the data warehouse. |
| 1.2 | 3/4/2003 | IBI Dev | Association tables were added to SeETL[RT]. The User Guide has been updated to reflect this new feature of SeETL[RT]. |
| 1.3 | 22/1/2004 | IBI Dev | Addition of Appendices explaining data warehouse data modelling in more detail. |
| 1.4 | 25/3/2004 | IBI Dev | Consolidation of the SeETL[RT] Utilities into SeETL[RT] User Guide.<br>Addition of:<br>• Delimiter Separated Values Reformat Utility<br>• Fixed File Format Reformat Utility<br>• Load Interface File option of the Data Transfer Utility |
| 1.5 | 2/4/2004 | IBI Dev | 1. Addition of the documentation for the Generate Delta File Utility. |
| 1.5.4 | 20/5/2004 | IBI Dev | 1. Bring the version number of the documentation into line with the version number of the software.<br>2. Add the first released of the scheduler to the documentation.<br>3. Add the first release of the DataStage batch processing utility to the documentation.<br>4. Update the error messages table with the new error messages for the new utilities.<br>5. Allow the Data Transfer Utility to end with a zero return code even if there is no input data to the Data Transfer Utility. |
| 1.5.5 | 30/6/2004 | IBI Dev | See Appendix 3 – Enhancements for Version 1.5.5 |
| 1.6.1 | 1/1/2005 | IBI Dev | See Appendix 4 – Enhancements for Version 1.6.1<br>Appendices were split out from the full documentation in version 1.6.1. and they were also renumbered. |
| 2.0 | 1/1/2005 | IBI Dev | Documentation migrated to Instant Business Intelligence |
| 2.1 | 1/9/2005 | IBI Dev | Enhancements for SeETL[RT] 2.1.<br>See Appendix 6 – Enhancements for Version 2.1 |
| 3.0 | 1/6/2006 | IBI Dev | Enhancements for SeETL[RT] 3.0.<br>See Appendix 7 – Enhancements for Beta Version 3.0 |
| 3.0.00 | 1/4/2007 | IBI Dev | Enhancements for SeETL[RT] 3.0.00<br>See Appendix 8 – Enhancements for Version 3.0.00 Production Release. |
| 3.0.00 | 3/1/2008 | IBI Dev | More Enhancements for SeETL[RT] 3.0.00<br>See Appendix 8 – Enhancements for Version 3.0.00 Production Release. |
| 3.1.00 | 1/1/2012 | IBI Dev | More Enhancements for SeETL[RT] 3.1.00<br>See Appendix 9 – Enhancements for Version 3.1.00 Production Release. |
| 3.1.01 | 1/1/2013 | IBI Dev | More Enhancements for SeETL[RT] 3.1.01<br>See Appendix 10 – Enhancements for Version 3.1.01 Production Release. |
| 3.1.02 | 1/1/2014 | IBI Dev | More Enhancements for SeETL[RT] 3.1.02<br>See Appendix 11 – Enhancements for Version 3.1.02 Production Release. |
| 3.2.00 | 1/1/2019 | IBI Dev | More Enhancements for SeETL[RT] 3.2.00<br>See Appendix 12 – Enhancements for Version 3.2.00 Production Release. |

# 2. AUDIENCE

The intended audiences for the SeETL[RT] User Guide are:

- Technical developers working on a Star Schema Data Warehouse project using SeETL[RT].
- IT managers considering the purchase of SeETL[RT].
- IT managers considering the purchase of the BI4ALL Data Models.

# 3. INTRODUCTION

Firstly, we would like to say congratulations on acquiring SeETL[RT]!!!

We are sure you will find it an indispensable toolkit in the development of your star schema data warehouses. The previous versions of this code have certainly been of great use to our clients over the many years it has been in development. We at Instant Business Intelligence are very pleased to be able to bring our clients the SeETL suite of products including this Run Time component.

We are sure you will find it an indispensable toolkit in the development of your star schema data warehouses. The previous versions of this code have certainly been of great use to our clients over the years.

SeETL[RT] is a complementary and optional product that sits beside SeETL[DT] . SeETL[RT] predates SeETL[DT] as a product. SeETL[DT] was initially developed as the 'complementary product' to make implementations of SeETL[RT] faster and easier. This has happened to the point that SeETL[RT] is now 'optional'. The sql generation capability of SeETL[DT] can now deliver nearly 95% of what is delivered in SeETL[RT] . However, we believe there is still a long life for SeETL[RT] as there will always be people who need the ETL engine to sit on a hub and do work across the network. This is something that the SeETL[DT] sql generation capability is not going to be able to do any time soon.

When SeETL[RT] was written, in the early 00s, it was the next logical step in speeding up the implementation of star schema data warehouses. Rather than generating code, SeETL[RT] adapts processing at run time. This is in direct contrast with SeETL[DT] which generates SQL code at design time and is not capable of 'adaptation' at run time.

The features of this product are tried, tested, proven and were developed in the real world of implementing large star schema data warehouses over a number of years without the benefit of the ETL tools available at the time.

Instant Business Intelligence will bring more features and products to market around Business Intelligence. Each of these products will enhance the value of SeETL[RT] to you, our valued clients.

If you have purchased maintenance you will receive many updates to the functions that are available that will further reduce the time and effort you must spend in constructing your data warehouse.

This document will explain:

- What SeETL[RT] is.
- What SeETL[RT] is not.
- The general design philosophy behind SeETL[RT].
- The components of SeETL[RT].
- How to install and customise SeETL[RT] to generate you first star schema.

Having implemented your first star schema it is expected that you will know enough to use the product without further documentation.

We wish you the best of luck in your star schema data warehouse development project.

# 4. WHAT SEETL RUNTIME IS

SeETL$^{RT}$:

- Contains a set of parameters defining the processing required for a particular data warehouse implementation.
- Contains a suit of C++/ODBC 3.0 compliant programs built to load star schema data warehouses.
- 'Discovers' the data model from views of the underlying tables.
- Is complementary to the industry leading ETL tools if you have one of those.
- Enables customers to implement a star schema data warehouse without having to write any code to load it. In this it is unique. The best word to describe watching SeETL$^{RT}$ run is 'magic'. If you have been in data warehousing a while and are familiar with the amount of effort to load star schemas SeETL$^{RT}$ will occur as 'magic'.
- Is a way of implementing star schema data warehouses when you don't have an ETL tool available.
- Is a way of complementing your ETL tool to provide some functions that the ETL tools do not provide.
- Is implemented using Visual C++.NET and ODBC 3.0 compliant calls.
- Is written using ansi standard C++ not managed C++.NET. Therefore it runs on many unix platforms.

SeETL$^{RT}$ is executable code. It is not a code generator. This 'executable code only' environment is the major differentiator between SeETL$^{RT}$ and many other tools. Most tools generate some code and then compile that code or interpret that code at run time.

# 5. WHAT SEETL RUNTIME IS NOT

While it is important to describe what SeETL$^{RT}$ is, we believe it is just as important to explain what the SeETL$^{RT}$ is not.

SeETL$^{RT}$ is **not**:

- A generalised ETL development environment.
- A replacement for ETL tools.
  We believe in ETL tools and we have used a number of them. However, they come at a price and many companies implement data warehouses without the benefit of one of the ETL tools.

- A product that can implement complex transformations of data other than those specifically coded.
- A data modelling tool that will help you design your star schema data warehouse.

SeETL$^{RT}$ does **not**:

- Store any significant amount of metadata that is usable by other query tools.

  For example, SeETL$^{RT}$ does not allow you to store help text for fields that may be propagated to end user tools such as Business Objects. There are other tools on the marketplace which perform this function.

- Integrate to other tools available on the market place.

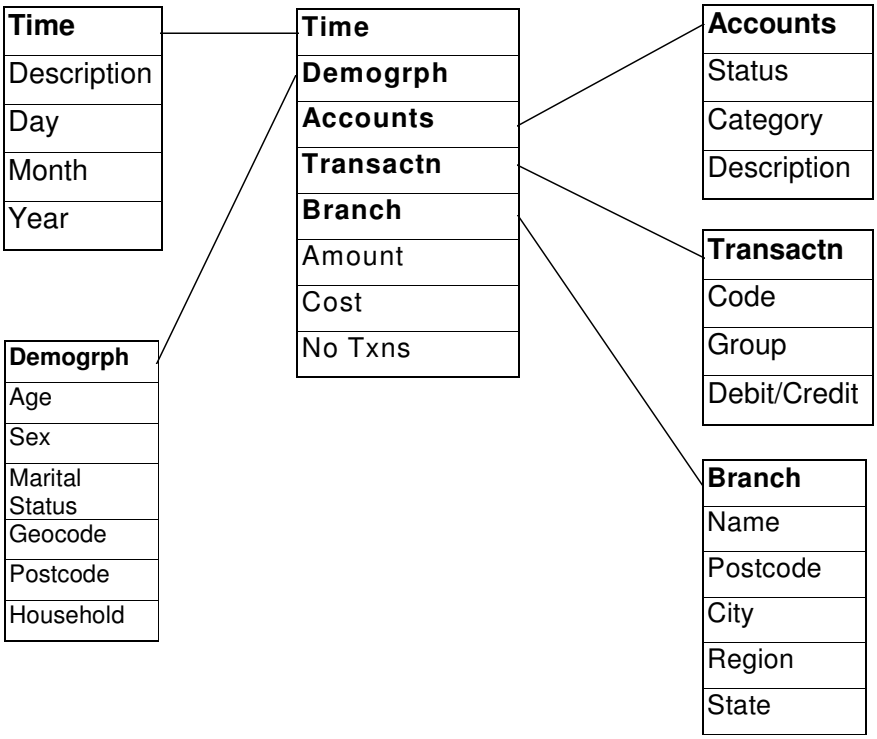
  The only available integration with other tools on the marketplace is that the programs are written in C++ and can therefore be called by any of the ETL tools as part of the ETL suite of programs.

Unlike most ETL tools the SeETL$^{RT}$ is **not** a development environment in itself. You can purchase the source code to the SeETL$^{RT}$ and use it to implement your data warehouse. In which case Microsoft Visual Studio C++.NET will be your development environment.

# 6. STAR SCHEMA MANAGEMENT

To explain what SeETL[RT] does let's briefly look at what a star schema looks like and how it might be built. There are many books on Star Schema modelling and it is suggested that the reader review some of these publicly available books for details on star schema development. SeETL[RT] is able to implement the most complex star schema models.

The following diagram is a simple star schema that might be useful for a banking business.




## 6.1. Input Record and Fact Tables

The first thing that we notice about star schemas is that they contain a basic central record which is stored in a 'fact table'. This input record could be a transaction record, or it could be a 'performance measures' fact record containing many performance measures for a particular period of time, usually weekly or monthly.

In SeETL[RT] fact table records are derived from the input record. The input record to be warehoused can contain any number of fields supported by your source and target database. Virtually all data types are supported. The exception is any data type that requires multiple calls to retrieve or replace the data. That is, graphics, blobs etc. These things are still very rare in data warehouses. If demand requires it they will be added.

The detailed level fact table should warehouse the entire input record, including the actual fields used as real keys which are translated to integer keys, as well as the integer keys required to join the detail fact record to the dimension tables. Where no record is found in the dimension table a key of '0' is recommended on the detailed fact record.

## *6.2. Dimension Tables*

The second thing we notice about a star schema is that there are dimension tables surrounding the central fact table. In SeETL[RT] you can create as many dimension tables as you like.

The only real limitation around dimension tables is how many tables can be joined in a single SQL statement. With most database managers this is 16. You can have many more than 16 dimension tables joined to a fact table as part of the data model, you just cannot actually join them all to the fact table in one query.

### 6.2.1. Summary Levels in Dimension Tables

Most people building a data warehouse are familiar with the concept of summarisation and understand that summaries are necessary to reduce the cost of running the data warehouse where large fact tables are being analysed or reported against.

For example, if you are a bank with 1M transactions per day and you want 3 years of detailed transactions available you are talking about 1 billion rows in your detail level fact table. Scanning this number of rows in order to determine transaction volumes in certain branches is a complete waste of computing resources.

If you have one or more large fact tables in your data warehouse you should strive to answer the question being asked using the minimum amount of computing resources required to actually answer the question. Summaries are the #1 tool you will have available to do this.

Summary data is actually only stored in the fact tables, however, the keys which are placed on the fact table must be stored somewhere and that somewhere is recommended to be the dimension tables. To understand how summary data is stored in a dimension table consider the time dimension. It is the most common dimension and the one most easily understood.

At the detailed level you will probably have the 'daily' level as the bottom level of the time dimension. So you will have rows as follows:

| Key | level | dim_chr_ky_fld | date | agg1 key | agg 2 key | agg3 key |
|---|---|---|---|---|---|---|
| 1 | detail | 2002-01-01 | 2002-01-01 | 1000001 | 2000001 | 9999999 |
| 2 | detail | 2002-01-02 | 2002-01-02 | 1000001 | 2000001 | 9999999 |
| 41 | detail | 2002-02-01 | 2002-02-01 | 1000002 | 2000001 | 9999999 |
| 42 | detail | 2002-02-02 | 2002-02-02 | 1000002 | 2000001 | 9999999 |
| 81 | detail | 2002-03-01 | 2002-03-01 | 1000003 | 2000001 | 9999999 |
| 82 | detail | 2002-03-02 | 2002-03-02 | 1000003 | 2000001 | 9999999 |
| 121 | detail | 2002-04-01 | 2002-04-01 | 1000004 | 2000001 | 9999999 |
| 122 | detail | 2002-04-02 | 2002-04-02 | 1000004 | 2000001 | 9999999 |
| 1001 | detail | 2003-01-01 | 2003-01-01 | 1000013 | 2000002 | 9999999 |
| 1002 | detail | 2003-01-02 | 2003-01-02 | 1000013 | 2000002 | 9999999 |
| 1041 | detail | 2003-02-01 | 2003-02-01 | 1000014 | 2000002 | 9999999 |
| 1042 | detail | 2003-02-02 | 2003-02-02 | 1000014 | 2000002 | 9999999 |
| 1081 | detail | 2003-03-01 | 2003-03-01 | 1000015 | 2000002 | 9999999 |
| 1082 | detail | 2003-03-02 | 2003-03-02 | 1000015 | 2000002 | 9999999 |
| 1121 | detail | 2003-04-01 | 2003-04-01 | 1000016 | 2000002 | 9999999 |
| 1122 | detail | 2003-04-02 | 2003-04-02 | 1000016 | 2000002 | 9999999 |

The records in the time dimension show:

- The first and second day of each month (and there would be a record for every day in each month)
- The level of 'detail' showing that these are the detailed dimension records.
- The dim_char_ky_fld which is a text string representing the date in the standard ISO date format which is the recommended format for strings representing dates.
- The real date in a date datatype. This can be any format you like to present to your users, however, the datatype is recommended to be 'date'
- The aggregate keys for level 1, 2 and 3. By extension you could consider how to build more levels.

For the purposes of this example we will use agg1 to be monthly, agg2 to be annual and agg3 to be 'total'.

Given this set of detail records, what might the aggregate level records look like? Consider the following table.

| Key | level | dim_chr_ky_fld | date | agg1 key | agg 2 key | agg3 key |
|---|---|---|---|---|---|---|
| 1000001 | monthly | 2002-01-01 | 2002-01-01 | 0 | 0 | 0 |
| 1000002 | monthly | 2002-02-01 | 2002-02-01 | 0 | 0 | 0 |
| 1000003 | monthly | 2002-03-01 | 2002-03-01 | 0 | 0 | 0 |
| 1000004 | monthly | 2002-04-01 | 2002-04-01 | 0 | 0 | 0 |
| 1000013 | monthly | 2003-01-01 | 2003-01-01 | 0 | 0 | 0 |
| 1000014 | monthly | 2003-02-01 | 2003-02-01 | 0 | 0 | 0 |
| 1000015 | monthly | 2003-03-01 | 2003-03-01 | 0 | 0 | 0 |
| 1000016 | monthly | 2003-04-01 | 2003-04-01 | 0 | 0 | 0 |
| 2000001 | annual | 2002-01-01 | 2002-01-01 | 0 | 0 | 0 |
| 2000002 | annual | 2003-01-01 | 2003-01-01 | 0 | 0 | 0 |
| 9999999 | total | 1900-01-01 | 1900-01-01 | 0 | 0 | 0 |

- The monthly level has a set of keys starting from 1000001.
- The yearly level has a set of keys starting from 2000001.
- The total level was given just a single key of 9999999.

This means as you perform your lookup of the detail level dimension table data for the time dimension you will be returned keys for the monthly, annual and total levels as well.

As you are well aware a very large amount of analytical requirements are at the monthly and annual level. You can minimise the computing resources to perform these monthly and annual analytical and reporting requirements by using these higher level keys in your summary fact tables.

You can access these aggregate levels of data in your summary fact tables by constraining the dimension table based on the 'level' column. For example, if you were after monthly transaction volumes through branches you would include "where time_dimension.level_col = 'monthly' " inside the query to select the data.

Some people recommend that the different levels of the dimension table should reside in physically separate tables. We, respectfully, disagree that this is necessary as placing views over the detailed level dimension table to show views of the dimension table at the various levels of summary available is trivial. And unless the dimension table is really large it causes very little performance overhead to access the summary level of the dimension table.

The exception to this recommendation is the Red-Brick database. The Red-Brick database has this wonderful feature that allows aggregate levels to be incrementally updated by the database loader based on the Referential Integrity constraints defined between the multiple levels of the dimension tables. Hence, in Red-Brick it is necessary to define the levels of data for different dimensions in different tables.

As you can see, if you select "where time_dimension.level_col = 'monthly' " the keys that will be returned from the time dimension are in the 1000001 series and hence the RDBMS will only look at those columns in the summary fact table that have time keys in the 1000001 series. This will greatly reduce the run time of the query as the monthly level of data should be around 1/30$^{th}$ the volume of the daily level transactions.

A similar situation exists for the 'annual' level. The volume of rows at the annual level should be at least 1/365$^{th}$ of the detail level. Hence, to produce a report at the annual level will quite likely search less than 2M records rather than 1B records. However, considering summaries may work together it is much more likely that a report looking at annual volumes of transactions would look at just a few tens of thousands of records rather than the 1B records.

By reducing the CPU consumption of trending queries by using summaries it becomes possible to perform much more complex queries with the same amount of investment in your hardware. In short, it pays to work smarter when designing your data warehouse.

Each dimension table can have up to 9 summary levels defined. The limitation is defined within the code for such things as array allocations etc. Though we see no reason as to why there would be any limitation on the code to have more levels of aggregation we have not tested with more than 9 aggregate levels. We can't imagine a case where a customer constantly needs more than 9 levels in a single dimension table.

If you did need more than 9 levels in a single dimension table you can always implement two dimension tables for the one dimension and then you will have 18 levels of summary available to you.

We have implemented the bottom level aggregate as 'detail' and the top level of each aggregate as 'total'. Today this is hard coded into the dimension table maintenance programs. In between the levels have been called 'level1', 'level2' etc.

The number of aggregate levels for a set of dimensions can be changed from star to star. You can vary the number of levels in dimensions from star to star by creating different views over the top of the underlying dimension tables. Remember SeETL$^{RT}$ should only ever see views of tables. Indeed, you can vary the number of aggregate levels visible in each dimension table for each fact table. They do not need to all have the same number of aggregates for the one star as is required in the C Data Warehouse Generator.

Take for example the 'time dimension' You may have one fact table where you rarely if ever want a summary level. This is the case for fact tables that are forecasts or budgets. Because they are so small you almost never want to provide summaries in the fact table itself. However, for the sales fact table you will always want many summaries across time because you will want to look at 'daily sales', 'weekly sales', 'monthly sales' etc. This is managed through different views of the time dimension table being used to look up the keys for the forecasts and budgets than when looking up the keys for the sales facts tables.

To accommodate this you can say that the 'sales budget' fact table has only 1 level of aggregate available in each dimension and the sales fact table has 9 levels available for each dimension.

Dimension tables are designed so that you can add any number of columns you would like. Thus, the dimension tables can contain anything that you want to select by. There are only a small number of required fields, all others are for selection and query processing.

One of the pieces of 'magic' in SeETL$^{RT}$ is that when you want to add more columns to the dimension tables you just add the columns to the underlying source/target tables and add the columns to the views. SeETL$^{RT}$ will move data from source to target by name with no intervention from your good self. This is a huge step forward for building star schema data warehouses.

**Number of Summary Levels in a Fact Table**

After giving it a significant amount of thought we have decided that a single detailed fact table can only aggregate into a single summary fact table and there can be many levels of aggregate in that summary fact table.

The previous cobol version of the code supported the ability to place N levels of summaries into M summary fact tables. This was because in very, very large summary fact tables for older versions of DB2 for OS/2 and OS/390 we found that splitting the summary fact tables to reduce the depth of the indexes helped performance. These problems have pretty much gone away and we have not used this feature for a number of years.

SeETL[RT] allows any number of summary levels from one single input fact table to be placed into a single summary fact table. Since each dimension table can contain up to 9 levels of aggregates the number of combinations of summaries is 10x10x10 for as many dimensions as there are attached to the fact table. So, for a 10 dimension fact table there are 10x10x10 etc (10,000,000,000) possible summary levels. We are certain that this many combinations of summaries will satisfy even the largest customers.

The way that aggregates are defined is by the 'aggregation_control' table. How to control summaries is documented under 'Components of SeETL[RT] '.

A point of note is that the primary key (integer) from the 'aggregation_control' table has been placed onto the summary fact table as the first column on the fact table. With the partitioning support available in SQL Server 2000, Oracle 9 and DB2 UDB it is entirely possible to physically partition the various aggregates in the summary fact table. This achieves the result that some people have felt was necessary, that being separate summaries should be in separate tables.

**Incremental Updates of Summaries**

Many Star Schema designs we have seen require that data is unloaded from the fact tables (or maintained on disk/tapes) and sorted/summarised/reloaded every time the Data Warehouse is run. This is a heavy CPU user. SeETL[RT] has been designed to always accept incremental updates using transaction files or period based snapshot files.

Thus, to maintain summary information the incoming transactions are consolidated with the data that exists in the fact tables. This dramatically reduces the CPU workload of maintaining summaries based on transaction files.

In the event that there are problems with the summarisation process or updates are backed out it is possible to rebuild all summaries from the detailed level fact table.

Something that you should be aware of when incrementally updating summaries is the problem of performing updates against a detailed fact record. If you update a detailed fact record and change the 'numeric' fields on the detailed fact record your detailed fact tables will no longer exactly match the summary fact table. If you must update detailed level fact records and you must also keep them in 100% agreement with the summaries you must generate two records, one to apply the negative of the adjustment and one to apply the positive of the adjustment so that the summaries remain in sync with the detailed fact records.

It is highly recommended that you do not adjust detailed fact records. It is recommended that you insert new records to make adjustments.

Having said that, there are some specific types of applications where adjusting the detailed level fact record is highly desirable. One example being the newspaper/magazine publishing business where returns data is best stored on the same record as the distribution record. The returns data does not come back until some significant period after the distribution record has been created.

# 7. COMPONENTS OF SEETL RUNTIME

This section describes the components of SeETL[RT].

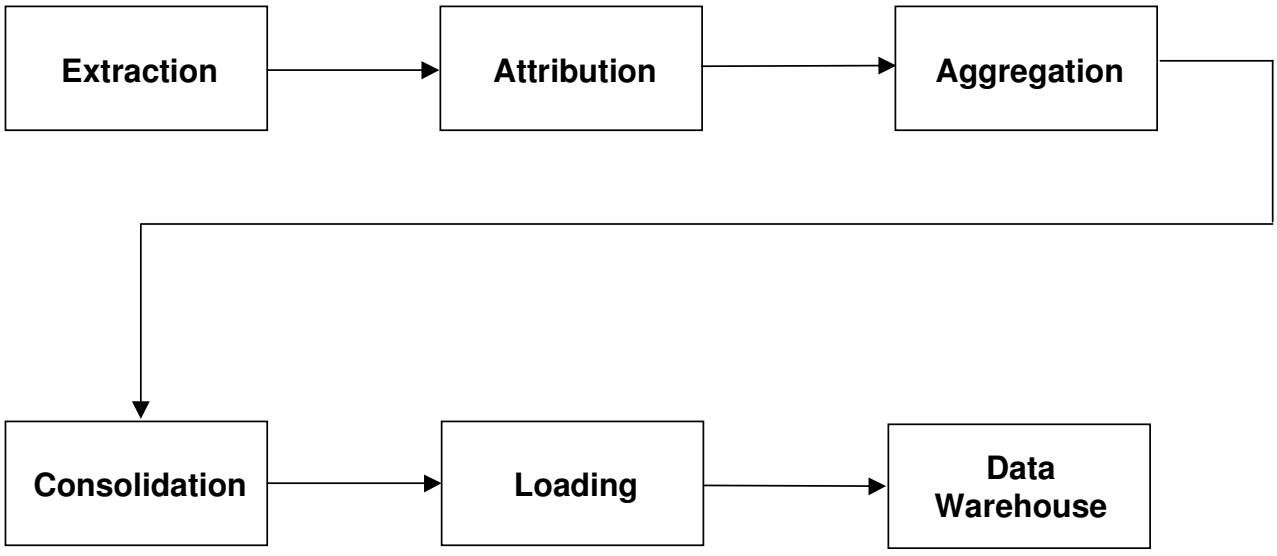
There are very few components for the executable version. The classes and source code documentation is provided separately.

## 7.1. Program Processes

There are currently 23 executable programs available which make up SeETL[RT] and associated utilities. They support the general processing of placing data into a star schema data warehouse:

- **Extraction** – from some upstream source be that an operational system, a staging area or any other form of upstream system.
- **Attribution** – The process of translating real character keys from upstream systems to integer keys for usage within the data warehouse.
- **Aggregation** – The process of aggregating just the transactions or records being loaded into the data warehouse for this run.
- **Consolidation** – The process of comparing the summary records produced by this incremental run with the summary records already stored in the data warehouse. The output of which is a file which can apply inserts and updates to the summary fact table.
- **Loading** – The process of loading detailed records into the detailed fact table and the updates/inserts for the summary fact table.
- **Type 1 Dimension Maintenance –** The process of loading data into type 1 dimension tables.
- **Type 2 Dimension Maintenance –** The process of loading data into type 2 dimension tables.
- **Association Maintenance –** The process of maintaining associations between dimensions. This is not depicted in the diagram below.

Note that the Extraction and Loading Processing is actually performed by the one program which is CTLU001 the Data Transfer Utility.



The following table describes the programs supplied. They are documented in their processing order.

| Program Name | Program Description |
|---|---|
| CTLDM01 | This is the type 1 dimension table processing program. |
| CTLDM11 | This is the type 1 dimension table processing program for initial loading. |
| CTLDM02 | This is the type 2 dimension table processing program. |
| CTLDM12 | This is the type 2 dimension table processing program for initial loading. |
| CTLAT01 | This is the attribution program. |
| CTLAG01 | This is the aggregation program. |
| CTLCL01 | This is the consolidation program. Current data types that can be 'consolidated' are the ODBC datatypes of:<br>SQL_INTEGER<br>SQL_SMALLINT<br>SQL_TINY_INT<br>SQL_DOUBLE<br>SQL_DECIMAL<br>SQL_REAL<br>Note that SeETL$^{RT}$ does not perform any validity checking to make sure that the consolidated field will not overflow the database target field. If the insert fails with a value that is too large then the database will trap the failure, pass a message to ODBC which will in turn print it to the error log and stop processing. Other data types may be added on demand.<br>Note that when using SQL_REAL as a data type for a field that will be consolidated the number of decimal places used defaults to 10. This is because the real data type does not define how many decimal places will be stored in the database. |
| CTLAS01 | The is the dimension table association processing program. It is an additional utility that can be run to create associations between dimensions. It is an optional program. |
| CTLBM01/02 | The Batch Maintenance Utility. A handy little utility that can act as a semaphore around batch processing. |
| CTLU001 | The Data Transfer Utility. A utility to move data between ODBC data sources. |
| CTLU002 | The SQL Statement Processor Utility. A utility that allows the user to run SQL commands using batch processing. Particularly handy for users who have to use many different database managers on a regular basis because it gives one consistent interface for all database managers. |
| CTLU003 | The SQL Server 2000 DDL Generator Utility.<br>A utility that takes a simple data model and generates SQL Server 2000 compliant SQL to create the tables and views. |
| CTLU004 | The Oracle 9 DDL Generator Utility.<br>A utility that takes a simple data model and generates Oracle 9 compliant SQL to create the tables and views. |
| CTLU005 | The Delimiter Separated Values Reformat Utility.<br>A utility that reformats Delimiter Separated Files into the Data Transfer Utility self describing file format. This effectively allows for the loading of Delimiter Separated Value files into the staging area to be further processed by SeETL$^{RT}$. In short, if you have data in systems that can only produce CSV/DSV files this utility can get this data into a format that can be loaded into an ODBC compliant database. |
| CTLU006 | The Fixed File Format Reformat Utility.<br>A utility that reformats Fixed File Format files into the Data Transfer Utility self describing file format. This effectively allows for the loading of Fixed Format Files into the staging area to be further processed by SeETL$^{RT}$. In short, if you have data in systems that can only produce Fixed Format Files this utility can get this data into a format that can be loaded into an ODBC compliant database. |
| CTLU007 | The Generate Delta File Utility.<br>A utility that accepts two files, old and current, and generates a third file of deltas. The delta file is the set of transactions that must have been applied to the old file in order to generate the new file. The Generate Delta File Utility is a generic piece of processing that allows the user to determine deltas between files at the file level before passing those files to SeETL$^{RT}$ for processing. |
| CTLU008 | The Batch Processing Scheduling Utility.<br>A utility that allows sophisticated suites of batches to be scheduled according to a variety of scheduling mechanisms. The Batch Processing Scheduling Utility replaces the standard command line interface used for the SeETL$^{RT}$ by placing the commands to run the SeETL$^{RT}$ into a table and executing those commands as part of a managed schedule. |

| | |
|---|---|
| CTLU009 | The Process Group Manager Utility.<br>A utility to manage groups of processes which comprise a batch which is run by the Batch Processing Scheduling Utility. |
| CTLU010 | The DataStage Job Submission Utility.<br>This is a simple utility to run datastage jobs from a command line interface or from within the Batch Processing Scheduling Utility. It provides an excellent mechanism for managing much more complex batches of DataStage jobs. |
| CTLU011 | The DataStage Job Submission Utility.<br>One of the more annoying problems of DataStage is the code that needs to be written to set parameters to run jobs and the fact that if you want to run jobs with different parameters you must have different files with different parameters passed to the jobs. It can be something of a 'pain' to be trying to find out which jobs run with which parameters. To overcome this problem we wrote a utility that starts DataStage job using parameters exposed to the job through a view that is passed the job submission utility. So, if some jobs require some special setting for their parameters it is trivial to set up a view to expose the parameters required for this specific job and to run it using those parameters.<br><br>This is something of an improvement on using files because the parameters are easier to change when they are in a table and they are VISIBLE to the developers without having to dig around through the DataStage code to find out which job is run with which parameters.<br><br>Lastly, because the DataStage job submission utility is submitted inside a 'database' environment there is now no need to change the names of the parameter files in the controlling DataStage job for a particular batch. A very cool utility!!! |
| CTLU012 | Load Dimension Tables into Memory Maps Utility.<br>This utility reads the ctl_dim_table_load_control table to determine if any dimension tables should be loaded into memory mapped files. For any table that should be loaded into a memory mapped file it will load the dimension table into a memory mapped file. It will overwrite any previously existing memory mapped file. |
| CTLU013 | MetaData Checking Utility.<br>SeETL[RT] is 'Typeless' and it does not check data types at run time. This is one of it's great features. If a data type can be converted to the target type by the ODBC driver it will be converted. However, experience in using SeETL[RT] has shown that one of the more difficult 'bugs' to find is where a field of the incorrect data type/contents is sent to another field. To help find these errors the MetaData Checking Utility inspects all source and target columns by name and issues messages where it detects that there might be a problem.<br><br>It has already proven to be a very useful utility. |
| CTLU014 | MetaData Printing Utility.<br>One of the real 'missing features' of most databases is that it is not easy to query their catalogs to determine the data types of columns in views. This is quite frustrating. SeETL[RT] makes extensive use of views and often times it would be very handy to just have a printout of the data types of the views. The MetaData Printing Utility prints the metadata for a single table to a single work file in the self describing file format used for other data. It is easy to read and has come in very handy. |
| CTLU015 | MetaData Loading Utility.<br>The MetaData Printing Utility proved such a hit the next question we were asked was whether we could create a utility to load the MetaData for a view into a table, rather than just print it. Of course, this was trivial and so the MetaData Loading Utility was created. This utility allows a user to load the column definitions for a view/table into a working table. The working table defaults to the name **ctl_column_defs.** However, the user can load the MetaData to any target table that meets the format of the table defined in the documentation. |
| CTLU016 | Data Correction Utility.<br>SeETL[RT] Utilities were planned not to contain data correction routines as these routines can be very customer/data specific. However, one of our major clients needed data correction routines and so we have included them as a separate utility. This currently corrects the formatting of numeric fields and dates into standard formats that can be accepted by ODBC drivers. It is expected more data correction facilities will be included over time. |
| CTLU017 | Batch Sleep Utility.<br>Many implementations require the scheduler to 'sleep' for a period. Rather than use operating system 'sleep' commands we created a SeETL[RT] utility that simply sleeps for the number of seconds the architect tells it to using a parameter. |
| CTLU018 | Simple XML Data Reader. |

| | |
|---|---|
| | Many implementations require that XML files be read. This utility reads simple XML files and reformats them into delimited files so they can be loaded into standard relational databases. |
| CTLU019 | Ready To Be Loaded Record Deletion Utility.<br>In many cases using the database loader is faster then using ODBC Insert statements. Usually the gap is about 5 times but it can be larger. In cases where it is warranted the Architect will wish to delete the rows that have been prepared for loading by the batch processing. This program takes a load file and performs deletes of all the rows in the file. These deletes have been prepared in a prior step in the processing. This is very useful when there are only a small number of updates in very large files. These types of condition commonly occur in telephone companies and their CDR records. The insert to update ratio is 9999+ : 1. |
| CTLU020 | SQL Server Bulk Load Utility<br>This program was written to place a 'wrapper' around the SQL Server Bulk Load Utility. Previous clients had called the bulk load utility directly from the scheduler. But a new client wanted the call to the bulk loader to be wrapped in a SeETL$^{RT}$ Utility to make for better audit capability. As ever we were happy to oblige. |
| CTLU021 | Start a SeETL$^{RT}$ Schedule Command Utility.<br>When performing testing it has previously been required that the commands that were inside the schedule tables would also need to be copied to files to be able to test one command at a time. This duplication was seen as 'dangerous' in that during alterations to a production system the commands in the files might not be copied back into the schedule. To eliminate the risk of this mistake the client asked for a command that fired the command inside the schedule as a 'one off' command that could be execute from the DOS box in windows. This is trivial so we wrote the utility. |
| CTLU022 | The Batch Processing Scheduling Utility.<br>This is a near duplicate of CTLU008. It is a utility that allows sophisticated suites of batches to be scheduled according to a variety of scheduling mechanisms. The Batch Processing Scheduling Utility replaces the standard command line interface used for the SeETL$^{RT}$ by placing the commands to run the SeETL$^{RT}$ into a table and executing those commands as part of a managed schedule.<br>The difference with this utility is that it connects to the network each cycle of the schedule every 'BacthSleepSeconds'. This is to increase the resilience of the scheduler. In implementations where the scheduler runs on a 'hub' (not recommended) any 'flicker' in the network causes the scheduler to stop and need to be restarted. This version of the scheduler uses more resources but is more resilient. It is the version of the scheduler we recommend when SeETL$^{RT}$ is implemented as a hub. |
| CTLU023 | SQL Statement Execution Utility.<br>This program is a relatively simple utility that reads SQL that accepts a parameter called "SQLFilename" and reads the SeETL Dictionary to find the relevant piece of SQL. It then submits it to the database manager and logs the start/stop timestamp, the number of rows processed and the first 8,000 bytes of the statement executed.<br><br>This is part of the upgrade to make the execution of the SQL snippets created by SeETL$^{DT}$ more secure, robust, reliable and auditable. |
| CTLSG01 | The PCS Segmentation Data Creation Program.<br>In order to make the 'rules' for the Product Catalogue Segmentation processing as flexible as possible it was determined that the 'rules' would be placed into views over the ***BI4ALL Data Model*** Tables. This is the 'ultimate' in flexibility since it means anything can be done.<br>This program then reads these views and unloads the data to a file ready to be loaded by subsequent programs. The unload/load style of processing was used because on SQL Server the run times for this program are very, very long. The views read large numbers of sales transactions in order to calculate the points. |
| CTLSG02 | The PCS Segmentation Data Pivot Program.<br>The prior program unloads data in long rows with 64 segment columns, one column for each segment, as well as a total column for the total segment. This data is then pivoted to have one row per segment where the segment score is non-zero. This pivoted data is then loaded into a separate table for the segmentation engine that will allow reporting on the rows more easily than the long rows with 65 columns representing the segment scores. |
| CTLSG03 | The PCS Segmentation Statistics Data Preparation Program 1<br>A number of statistics are being generated on the data that resides in the ***BI4ALL Data Models***. The plan is to build more calculation engines in the future. This program sums and counts instances of segment scores in preparation for the calculations on this data in the next program. |
| CTLSG04 | The PCS Segmentation Statistics Calculation Program 1 |

| | |
|---|---|
| | The prior program prepares data into temporary tables that are then used by this program to calculate to co-efficiency of co-relation between the various segments in terms of strength of co-relation compared to other segments. This is the first true calculation engine placed on top of the ***BI4ALL Data Models***. |

To control the processing of each program a set of parameters are passed to the program. These parameters will be discussed in detail later in this document.

## 7.2. Tables Required in the Target Database

This section documents the tables required in the target data warehouse database. All permanent parameters are stored in the target data warehouse database.

## 7.3. Message Tables

It is highly recommended that you write error messages and audit messages to the data warehouse database. To do so two tables are required to exist in the target database to receive the messages. These are the ctl_message_table and the ctl_audit_table. Example DDL for each of these tables is provided below.

### 7.3.1. ctl_audit_table

The ctl_audit_table contains the following fields.

| Field Name | Description |
|---|---|
| program_name | The program name that issued the write of the audit/control record to the database. |
| field_name | The name of the field that was audited or a short message to tell you what file record count was reported on this record. |
| total_rows | The total number of rows processed via this file for the run. |
| integer_total | The hash total for an integer field that was audited as a 'critical field' |
| dollar_total | The hash total for a dollar field that was audited as a 'critical field'. |
| started_tstamp | The date time that the program was started using a timestamp. |
| ended_tstamp | The date time that the audit record was written to the database using a timestamp. |

Example DDL for the table is as follows:

```
create table dbo.ctl_audit_table
    (  program_name      varchar     (0256)       not null   default 'unknown'
    ,  field_name        varchar     (0256)       not null   default 'unknown'
    ,  total_rows        integer                  not null   default 0
    ,  integer_total     integer                  not null   default 0
    ,  dollar_total      money                    not null   default 0
    ,  started_tstamp    datetime      not null   default current_timestamp
    ,  ended_tstamp      datetime      not null   default current_timestamp
    )
;
```

### 7.3.2. ctl_message_table

The ctl_message_table contains the following fields.

| Field Name | Description |
|---|---|
| program_name | The program name that issued the write of the error message to the database. |
| function_name | The name of the function inside the program that issued the write of the error message to the database. |
| message_number | The error message number. This can be used to look up error message documentation and to report to the technical support for the product. |
| message_severity | The severity of the message. There are three levels:<br>• I – Information. The message is provided for information only and can generally be ignored. Examples of Information Messages are 'Program Starting.' 'Program Ending.'<br>• W – Warning. A condition has been found that is unusual and unexpected but processing is not stopped. All warning messages should be investigated.<br>• S – Severe. A condition has been found that is unexpected and processing will be stopped. Examples are where the ODBC connection parameters do not result in a connection being established, input or output tables not being found etc. |
| message_text | The text of the message. Messages are issued to be as descriptive as possible. |
| date_time | The system date time that the error message was issued. |

Example DDL for the table is as follows:

```
create table dbo.ctl_message_table
    (  program_name      varchar     (0256)    not null default 'unknown'
    ,  function_name     varchar     (0256)    not null default 'unknown'
    ,  message_number    varchar     (0010)    not null default 'unknown'
    ,  message_severity  varchar     (0001)    not null default 'unknown'
    ,  message_text      varchar     (4000)    not null default 'unknown'
    ,  date_time         datetime               not null default current_timestamp,
    )
;
```

## 7.4. Control Tables

This section documents the control tables that are required to be in the database and set up when running SeETL[RT].

### 7.4.1. ctl_last_key_used

The ctl_last_key_used table is used to store the last key used for each dimension in the same database as the dimension table updates occur.

The ctl_last_key_used table contains the following fields.

| Field Name | Description |
|---|---|
| pk_int_key | This is the primary key of the table. It is an integer that you can enter via some tool such as MS Access. You only ever need to enter it once per dimension table so no front end has been developed. |
| dim_table | The name of the 'dimension table' for which this record stores the last key used. This can be any valid database table or view name. It is recommended that it is the name of a view. Examples are 'time_dim', 'customer_dim' etc. |
| last_key_used | This will store the last key used for a particular dimension table. You only ever enter the key at startup time for the data warehouse and it is not recommended that it is ever changed. Integer keys will be allocated sequentially for each dimension table. We prefer not to use database specific sequence keys as they can often impede unload/load activities on tables by insisting on providing new keys for records being inserted. |

Example DDL for the table is as follows:

```
create table dbo.ctl_last_key_used
    (  pk_int_key       integer                 not null primary key
    ,  dim_table        varchar     (255)       not null default 'unknown'
    ,  last_key_used    integer                 not null default 0
    )
;
```

Example rows for this table are as follows:

| pk_int_key | dim_table | last_key_used |
|---|---|---|
| 1 | time_dim | 1 |
| 2 | product_dim | 1 |
| 3 | customer_dim | 1 |

These records indicate that the three dimension tables will start numbering their new records from '2'.

### 7.4.2. ctl_aggregation_control

One of the major features of SeETL[RT] is the ability to generate new aggregates without any program changes, merely a parameter change in this table.

The ctl_aggregation_control table contains the following fields.

| Field Name | Description |
|---|---|
| pk_aggregate_number | This is an integer primary key to this table. It is actually passed through to the summary fact table to provide the ability to partition the summary fact table by the individual summary. It can also be specified in a view to define the fact table as a separate table to end user tools such as business objects and microstrategy. |
| run_type | The type of run for this aggregation. Summaries which you want to occur for every aggregation run of the fact table should be marked as 'always'. Aggregation levels that you want to generate just once, perhaps for special analysis, you should mark as 'oneoff'. |
| fact_table_name | The name of the detailed level fact table that you want the summary to be generated from. An example is f_order_fact. The summary table will be assumed to be 'f_order_fact_summary'. Thus you should choose naming standards for fact tables that allow the suffix '_summary' to be added to the table name. |
| number_of_dimensions | The number of dimensions in this fact table. This field is used to select the correct number of columns from the level_dimension_n fields. For example if there are 4 dimensions, only the first 4 level_dimension fields will be read as significant. |
| level_dimension_1 | The level of the dimension to be used in this summary level. For example 0 means 'detail' level, 1 means 'level1', 2 means 'level2' etc. You have one level field for each dimension for each aggregation for the fact table. This means that the level key for that level will be used in the summary fact table. You can have many levels of aggregation and hence for the time dimension you can construct levels at daily, weekly, monthly, quarterly, annual and total levels if you choose to. |
| level_dimension_2 | The same explanations as level_dimension_1 except this is for the second dimension on the fact table. |
| etc. | |
| level_dimension_50 | The same explanations as level_dimension_1 except this is for the fiftieth dimension on the fact table. |

Please note there are a maximum of 50 dimensions allowed for any single fact table. If anyone has a need to go over this please let us know. The code is easy to change.

Example DDL for the table is as follows:

```
create table dbo.ctl_aggregation_control
    (  pk_aggregate_number     integer          not null  primary key
    ,  run_type                varchar    (20)  not null  default 'always'
    ,  fact_table_name         varchar (256)   not null  default 'unknown'
    ,  number_of_dimensions    integer          not null  default 0
    ,  level_dimension_1       integer          not null  default 0
    ,  level_dimension_2       integer          not null  default 0
    ,  level_dimension_3       integer          not null  default 0
    ,  level_dimension_4       integer          not null  default 0
    ,  level_dimension_5       integer          not null  default 0
    ,  level_dimension_6       integer          not null  default 0
    ,  level_dimension_7       integer          not null  default 0
    ,  level_dimension_8       integer          not null  default 0
    ,  level_dimension_9       integer          not null  default 0
    ,  level_dimension_10      integer          not null  default 0
    .
    .
    .
    ,  level_dimension_45      integer          not null  default 0
    ,  level_dimension_46      integer          not null  default 0
    ,  level_dimension_47      integer          not null  default 0
    ,  level_dimension_48      integer          not null  default 0
    ,  level_dimension_49      integer          not null  default 0
    ,  level_dimension_50      integer          not null  default 0
    )
;
```

Example rows for this table are as follows:

| pk aggregate number | run type | fact table name | number of dimensions | level dimension 1 | level dimension 2 |
|---|---|---|---|---|---|
| 1 | always | f_order_fact | 2 | 2 | 1 |
| 2 | always | f_order_fact | 2 | 5 | 0 |
| 3 | always | f_order_fact | 2 | 1 | 5 |

These rows mean that the f_order_fact_summary table will contain 3 aggregate levels. There are only 2 dimensions in the fact table.

The first aggregate will summarise the first dimension table at the key level 2 and the second dimension at the key level 2.

The second aggregate will summarise the first dimension table at the key level 5 and the second dimension at the detail level key.

The third aggregate will summarise the first dimension table at the key level 1 and the second dimension at the key level 5.

How these keys get allocated will be explained below in ctl_dim_table_key_definitions.

### 7.4.3. ctl_dim_table_key_definitions

One of the major features of SeETL[RT] is the ability to generate multiple levels of aggregates depending on information fed into the dimension table. These levels in the dimension tables can then be used to generate levels in the summary fact tables. In all other solutions we have ever seen, this must be done in code. A new aggregate level being made available in the dimension table means someone has to make a change to some code somewhere. No more!!

The ctl_dim_table_key_definitions table contains the following fields.

| Field Name | Description |
|---|---|
| pk_int_key | This is an integer key to the table that you must enter. |
| dim_table_name | This is the dimension table name. For example time_dim, customer_dim, product_dim etc. |
| dim_type | This is the 'type' of the dimension, type 1 or 2. Valid values are 1 or 2. |
| agg_level | This is the aggregate level that the set of concatenated columns will represent. Valid values are 0-9. 0 means 'detail' and 9 means 'total'. In between they are known as 'level1', 'level2', 'level3' etc. Note that there is no need for the levels to actually roll up into each other. For example level 0 could be day, level 2 could be week and level 3 could be month. And weeks do not roll up into months. |
| num_columns_joined | This is the number of columns that are to be joined together to form the string character key for this level. If you enter a lower number than fields you want to join SeETL[RT] will only join the first n columns. If you enter a higher number than the number of column names you provide it will only join on the number of column names available. However it is highly recommended that the number of columns joined accurately reflects the number of column names you actually want to join. The maximum is 50 joined columns. |
| column_names | This is the set of column names you want to join together to form the character key to be used at this level. Every unique combination of these characters will be allocated an integer key and placed into the dimension table marked as being at that level. If you join non character columns (such as integers) they will be converted to character strings and then concatenated. Note that trailing blanks in an input field will also be truncated. The concatenated string of characters is limited to 255 characters. This seems more than enough since we have never used more than 150 characters previously. The concatenation character is the '+' character.<br>Note that a column that is concatenated as part of the key that is null is allocated the internal character string 'NULL' and this string is placed onto the dimension record. This means that there is a restriction. It is recommended that columns used for dimension table keys that can be null should not have the real value of 'NULL' as a possible value. |

Example DDL for the table is as follows:

```
create table dbo.ctl_dim_table_key_definitions
    (  pk_int_key             integer                 not null primary key
    ,  dim_table_name         varchar        (256)    not null default 'unknown'
    ,  dim_type               integer                 not null default 0
    ,  agg_level              integer                 not null default 0
    ,  num_columns_joined     integer                 not null default 0
    ,  column_names           varchar        (4000)   not null default 'unknown'
    )
;
```

Example rows for this table are as follows:

| pk int key | dim table name | dim type | agg level | num columns joined | column names |
|---|---|---|---|---|---|
| 0 | customer_dim | 1 | 0 | 1 | cust_code |
| 1 | customer_dim | 1 | 1 | 1 | cust_household |
| 2 | customer_dim | 1 | 2 | 2 | cust_sex+cust_geocode |
| 3 | customer_dim | 1 | 3 | 1 | directive_none |
| 4 | customer_dim | 1 | 4 | 1 | directive_none |
| 5 | customer_dim | 1 | 5 | 1 | directive_none |
| 6 | customer_dim | 1 | 6 | 1 | directive_none |
| 7 | customer_dim | 1 | 7 | 1 | directive_none |
| 8 | customer_dim | 1 | 8 | 1 | directive_none |
| 9 | customer_dim | 1 | 9 | 1 | Directive_total |
| 10 | time_dim | 1 | 0 | 1 | day_field |
| 11 | time_dim | 1 | 1 | 2 | week_number+year_field |
| 12 | time_dim | 1 | 2 | 2 | month_number+year_field |
| 13 | time_dim | 1 | 3 | 1 | directive_none |
| 14 | time_dim | 1 | 4 | 1 | directive_none |
| 15 | time_dim | 1 | 5 | 1 | directive_none |
| 16 | time_dim | 1 | 6 | 1 | directive_none |
| 17 | time_dim | 1 | 7 | 1 | directive_none |
| 18 | time_dim | 1 | 8 | 1 | directive_none |
| 19 | time_dim | 1 | 9 | 1 | directive_total |

The data in these rows has the following meaning.

Both customer_dim and time_dim have 1 detail level and 9 aggregate levels. You do not need to have 9 aggregate levels. However, if you want to leave spaces for possibly having more levels in the future you must enter the rows for each level you might want to have at some point in the future. SeETL$^{RT}$ assumes that the first time it reads this table it is reading the maximum number of levels you will ever want for this dimension. Increasing the number of levels at a later date is not tested and not supported.

Aggregate level '0' means the 'detail' level.

The text 'directive_none' has a special meaning. It means that this level of aggregate is reserved for future use but is not currently implemented.

The text 'directive_total' has a special meaning. It means that this level of aggregate is the highest level of aggregate and means total for that dimension. Hence, when you want to ignore a dimension in a summary you can set it to the total level and all rows in the summary level will have the same key.

The concatenation character '+' has special meaning in that it breaks the column names. The code looks at the '+' sign in this table as the delimiter of the field names. Naturally '+' is not allowed as a character in the table column name.

The concatenation of week_number+year_field means that when the dimension table is loaded these two fields will be joined together and the dimension table will be searched to see if this combination of week and year has ever been loaded. If not, it will allocate a key and insert the record thus creating a unique character string for week level and an integer key that can be used in the fact table to join to the dimension table record.

This processing is for type 1 dimensions and for the keys for type 2 dimensions. However, type 2 dimension tables have a further requirement to close out records and insert new records when a field that is not necessarily part of the key changes. Naturally, any field that changes in the key will insert a new record if it does not exist. To handle type 2 dimension tables another control table is required. It is discussed next.

### 7.4.4. ctl_dim_table_type2_col_defs

To support type 2 dimension table changes we must record the combination of fields that must be checked for change to determine if the existing dimension table record should be closed out and a new type 2 record should be inserted. The ctl_dim_table_type2_col_defs table supports this requirement.

The ctl_dim_table_type2_col_defs table contains the following fields.

| Field Name | Description |
|---|---|
| pk_int_key | This is an integer key to the table that you must enter. |
| dim_table_name | This is the dimension table name. For example time_dim, customer_dim, product_dim etc. |
| num_columns_to_test | This is the number of columns that are to be tested to see if they have changed when compared to the current dimension table record. There is a limit of 50 fields. |
| column_names | This is the set of column names you want to test for change. Each column's data value on the incoming record will be compared with the value on the current record for the dimension. The comparison is null sensitive so if the values are both equal or they are both null the record will be considered not to have changed. The delimiter is again the '+' character. |

Example DDL for the table is as follows:

```
create table dbo.ctl_dim_table_type2_col_defs
    (  pk_int_key             integer                 not null primary key
    ,  dim_table_name         varchar        (256)    not null default 'unknown'
    ,  num_columns_to_test    integer                 not null default 0
    ,  column_names           varchar        (4000)   not null default 'unknown'
    )
;
```

An example row for this table is as follows:

| pk int key | dim table name | num columns joined | column names |
|---|---|---|---|
| 1 | customer_dim2 | 3 | cust_addr_1+cust_addr_2+cust_phone |

The data in this rows has the following meaning.

The customer dimension 2 is a type 2 dimension table. The fields to check (at the detailed level only) for change are cust_addr_1, cust_addr_2 and cust_phone. If any of these fields have changed then close out the current dimension record and open a new dimension record. The source of the dates for record closure and opening is discussed below.

### 7.4.5. ctl_dim_table_load_control

Fact table attribution processes have a 'need for speed'. There are two Fact Table Attribution programs provided with SeETL[RT]. One is the standard processing program and one is for special processing required for the Sybase IWS data model. The attribution process can consume very large amounts of processing power and run for very long elapsed times.

One of the standard ways to speed it up is to trade memory for speed. That is load the dimension tables into memory and search them there rather than read them through the database which is at least 10 times slower per read. This loading of dimension tables into memory is facilitated by the ctl_dim_table_load_control table.

The ctl_dim_table_load_control table contains the following fields.

| Field Name | Description |
|---|---|
| pk_int_key | This is the integer key to this record. |
| fact_table_name | This is the name of the fact table for which this record specifies the dimension table load option.<br>**(2.1 Enhancement).** The fact table name can also be set to '*ALL*' to specify that the dimension table named in dim_table_name should be loaded as specified in the load_control_option field for all fact tables. |
| dim_table_name | This is the name of the dimension table for which this record specifies the dimension table load option. The reason that both the fact table and the dimension table are specified is so that you have the option of loading dimension tables into memory for specific fact tables. The larger fact tables will benefit more from loading dimension tables into memory. |
| load_control_option | Determines whether the dimension table will be loaded into memory and searched using a binary search or read from the database. Valid values are:<br>1. Read the table from the database.<br>2. Read the table from a 100 element array. If the row is not found then read it and put it into the array.<br>(Note. This option operates exactly as Option 1 at this time. With memory mapped IO implemented this option may not be implemented in the future.)<br>3. Load the table into memory and read the dimension table from memory using a binary search.<br>4. Load the table into a memory mapped file backed by the system paging file for windows 2000 and by /tmp files on unix. If a dimension table is loaded into a memory mapped for by one attribution process (AT01/2) it is available for access by all attribution processes. Memory mapped IO has been added to provide 'unlimited scalability'. That is, large volumes of memory can be used to load one single copy of as many dimension tables as the user would like and these images are shared across as many processors as the user would like to use. |
| dimension_table_group | **(2.1 Enhancement).** This field can be defined to be any string, though we do recommend 'GROUP1', 'GROUP2', GROUPn' etc. With the introduction of memory mapped IO we also introduced the ability to run many instances of CTLU012 at the same time and to load many dimension tables at one time. This is made possible by specifying groups of dimension tables to load into memory mapped files. The dimension tables that are to be loaded in a single group are defined using this field. They are loaded by CTLU012 using the `LoadDimensionTableGroup` parameter.<br><br>By using this field it is possible for the user to gain the maximum amount of control over the loading of dimension tables into memory mapped files enabling the most efficient use of the machine resources to achieve maximum throughput of the overall batch while using the minimum amount of real memory to maintain copies of the dimension tables in the memory of the machine. |

Example DDL for the table is as follows:

```
create table dbo.ctl_dim_table_load_control
    (  pk_int_key            integer                 not null primary key
    ,  fact_table_name       varchar        (256)    not null default 'unknown'
    ,  dim_table_name        varchar        (256)    not null default 'unknown'
    ,  load_control_option   integer                 not null default 0
    ,  dimension_table_group varchar        (256)    not null default 'unknown'
    )
;
```

Example rows for this table are as follows:

| pk int key | fact table name | dim table name | load control option |
|---|---|---|---|
| 0 | f_order_fact | customer_dim | 1 |
| 1 | f_order_fact | time_dim | 2 |
| 2 | f_order_fact | product_dim | 3 |
| 3 | f_order_fact | product_dim | 4 |

The data in these rows has the following meaning.

For the f_order_fact attribution process CTLAT01/02 will:

1. Read customer_dim and time_dim from the database

2. Load product_dim into in an in memory array and binary search the array when looking up product keys.

3. Load vendor_dim into a memory mapped in memory array and binary search that array. The vendor_dim table will also be available for any other instance of CTLAT01/02 to share while product_dim is private to the CTLAT01/02 process that loaded it.

4. Note that no record in this table for a fact table/dimension table combination is equivalent to specifying a load control option of 1, read the table from the database.

#### *7.4.5.1. Guidelines on In Memory Array Allocations*

The use of the in memory arrays is important enough to provide some advice and guidance on their use.

The allocation of these arrays has been built to be as efficient as possible in the use of memory so that it is possible to load the greatest amount of data from dimension tables into memory.

The dimension character string key field for each record only has as much memory allocated as the length of the actual string on the database record, plus a (char *) pointer to point to the string.

This innovation allows:

1. The dimension tables to have dimension character string keys defined as 255 character strings rather than some fixed limit. Having longer character strings for some dimensions is particularly useful.

2. For character strings of significant variation in length to be used in the one dimension table. This is very useful for multiple levels being built into dimension tables.

3. For the minimum use of memory for any particular set of character keys stored in a dimension table.

The integer keys are allocated as one large array that contains (number of records in the dimension table x number of integer keys per record). This single array is then navigated to find the set of keys required when the character string key is found.

The search mechanism is a binary search. Though it would be faster to use a hash search we have not done so. Should customers require hash searching of the dimension tables it can be implemented in a later release.

**Calculating Memory Consumption**

When determining which dimension tables to load into memory you should consider the following information which gives an indication of memory consumption.

Character keys memory requirement = (Number of rows x 4bytes) + Actual volume of string data.

Integer keys memory requirement = (Number of rows x Number of integer keys/row x sizeof (integer) bytes).

All memory for all dimensions must be able to be allocated for the entire life of any instance of CTLAT01/02.

For example if you load a customer dimension which has 9 aggregate levels and 1 million rows with an average character string length of 50 bytes you will require memory as defined below.

| | | |
|---|---|---|
| Character keys memory requirement | = | (1000000x4)+(1000000x51) |
| | = | 4MB + 51MB |
| | = | 55MB |
| Integer keys memory requirement | = | 1000000x9x4 |
| | = | 36MB |
| Total Memory that will be allocated | = | 91MB |

If four instances of CTLAT01 are run at the same time and they all load the customer dimension into their own address space then the memory requirement will soon become quite significant. In general, now that memory mapped IO is available, large dimension tables should be loaded into shared memory and small tables should be loaded into private memory. Now that memory mapped IO is available only dimension tables which present keys in excess of **approximately 1.5GB (windows)** should need to be read directly from the database.

For 'flat' data models with character keys and only one integer key associated with them the average kind of row size one would expect is 30 bytes. **And 1.5 GB gives about 50M key rows**. This is enough for any company planning on running their DW on windows. On unix systems this approximate limit of 1.5GB is removed.

We will be impressed if any customers of SeETL[RT] have dimension tables with 1.5GB of keys!!

**Memory Mapped IO**

Memory mapped IO was added to the windows 2000 version of SeETL[RT] in version 1.6.1 and the unix version of SeETL[RT] in version 2.1. The memory mapped IO processing is separately licensable functionality for SeETL[RT] in the windows executable license of the software. It is 'always available' in the source code license version. The main purpose of introducing memory mapped IO is to remove the need to allocate large amounts of memory for multiple instances of the Attribution Program (AT01/02).

As described, memory mapped IO will load **one** copy of the dimension table into a memory mapped file and all Attribution processes will access that one copy saving large amounts of memory.

However, the user will benefit from understanding the various ways in which memory mapped IO can be used when setting up the batch process for SeETL[RT] in order to set up the batch stream most effectively.

There are two ways to have a dimension table loaded into memory:

1. Using the Load Dimension Tables into Memory Maps Utility (U012)

   or

2. Allow the Attribution process to load the dimension table into memory mapped IO.

These two methods have their advantages and disadvantages. If it is at all possible, it is ***highly recommended*** that the user use the Load Dimension Tables into Memory Maps Utility is used in ***all*** cases. However, it is acknowledged that it is possible SeETL[RT] might one day be used by a customer who has too much data for even one copy in memory to be in memory at the same time on a windows machine. And there is now a solution to this problem as well.

**Using the Load Dimension Tables into Memory Maps Utility**

The Load Dimension Tables into Memory Maps Utility scans the ctl_dim_table_load_control table and loads all dimension tables marked as being required to be loaded into memory mapped IO when it starts up. It does this whether those tables will actually be used by any Attribution programs in the current batch or not. The Load Dimension Tables into Memory Maps Utility does not know which tables **will** be used in the current batch just that the dimension tables should be loaded into memory.

The utility was written this way to allow SeETL[RT] to run with the minimum number of processes remaining resident during the batch processing. It would have been easy to write the utility so that one program only loaded one dimension table. This would also run a little faster because the loading of the dimension tables into memory could be parallelised. However, it would leave one process per dimension table running on the operating system for the duration of the Attribution processing. That could be a lot of processes for a long time. So it was decided to write the utility as just one program that loaded **all** dimension tables.

To do this there were features added to allow the Load Dimension Tables into Memory Maps Utility to be started before any Attribution process can start and to be stopped at the end of the batch. See the documentation for the Load Dimension Tables into Memory Maps Utility to see how this is implemented.

It is expected that no SeETL[RT] customer will ever have so many dimension tables that are so large that the cannot all be loaded into memory at the same time. However, it is possible, so a solution is provided.

**Allow the Attribution Process to Load the Dimension Table into Memory Maps.**

If the customer does not have the memory required to load all dimension tables they want to load into memory for the duration of the Attribution processing the Attribution programs contain a feature which will allow them to load the dimension table into memory mapped IO as and when the table is required. This means that only those tables that are required are loaded into memory. This further reduces the memory requirement for Attribution processing.

However, there is a cost associated with this approach. The user should be aware that for a memory mapped file to remain resident in memory a process must be using it. If no process is using it then it is discarded and must be loaded from the database again before it can be used. So a badly designed batch stream may re-load dimension tables a number of times.

Further, there is a level of communication required between processes to be able to know when a dimension table has been loaded into memory mapped IO. Because this situation is considered to be unlikely and there is an easy solution to the problem there is currently only minimal support for these communication requirements.

Those customers who have bought the source code can refer to the source code and the extensive comments in the source code to see exactly how this works.

What happens is as follows:

1. The Attribution Process detects that the dimension table is going to be used and detects that it should be loaded into a memory mapped file.

2. It asks the operating system if any such table is already loaded into a memory mapped file.

3. If not it reads the table to determine the size of the memory mapped table it needs to allocate.

4. It then allocates the memory mapped file and then re-reads the dimension table and loads it into memory. [1]

5. It then makes the memory mapped file available to other processes.

Of course, if two Allocation Processes arrive at the processor for starting at around the same time it is possible that they will both ask the question 'Is this dimension table loaded into memory?' at almost the same time and both get the answer 'no'. They would then ***both*** go ahead and read the dimension table twice and then both try to allocate a memory mapped file. One will allocate the file and the other one will get the answer 'memory mapped file already exists'.

In this case there will be a waste of reading through the dimension table twice for the second process. No harm is done and the processing works.

However, if the user is really unlucky and the the second process has very bad timing it might try to create the memory mapped file, get the answer that it is already there, and try to use it, while the first process is still loading it. There are currently no semaphores implemented to stop this unlikely but remotely possible situation.

To avoid this remote possibility the user should be aware of this situation and should make ***absolutely sure*** that there is enough time for the first attributiuon process to load the dimension table into memory before starting the second attribution process. If any number of users start to have this problem semaphores will be introduced into SeETL[RT].

If the first Attribution Process finishes before the second Attribution process the memory mapped files will remain in memory. In fact, the memory mapped files will remain in memory until the last Attribution Process has terminated.

Please note. This option is not supported on unix platforms as it is not expected that SeETL[RT] will ever be used on a large unix machine by a client so large that the dimension table keys cannot be loaded into the memory mapped io of the unix machine. In such cases we recommend 'buy some more memory'. We have only implemented this option to compensate for the limited memory supported by windows 2000+.

[1] To load a dimension table into a memory mapped file requires 2 reads of the complete dimension rather than just the one read of the dimension table required to load it into normal memory. This is because when loading the table into normal memory the memory for each string can be 'malloced' when the string is read. When reading into memory mapped files it is much faster to calculate the full size of the memory mapped file required before starting to load it.

### 7.4.6. ctl_batch_control

Those of us who have been around for a while remember that batches are processed on effective dates and that the system date is a poor guide as to the correct date to use on a record. So it is with type 2 dimension tables for the date_from and date_to columns. For example, if a data warehouse load fails on a Friday night and it will not be fixed until Monday the batches from Friday, Saturday and Sunday need to be processed. These batches actually contain changes from different days. Therefore it should be possible to place the correct date of change onto the type 2 dimension records. This is facilitated with the ctl_batch_control table.

The ctl_batch_control table contains the following fields.

| Field Name | Description |
|---|---|
| pk_batch_number | This is the number of the batch. It is a primary key for this table. |
| batch_date | This is the batch processing date for this batch. If batch processing has been delayed as in the example above this date can be changed to tell SeETL[RT] the processing date. As far as SeETL[RT] is concerned this date is the real date that applies to records being passed into it. SeETL[RT] uses batch_date – 1 day as the closing date for changed type 2 dimension records and batch date as the opening date for changed type 2 dimension records. |
| batch_complete_flag | This is a flag 0 or 1 to indicate that the batch is complete. When it is set to 0 the batch is in progress. SeETL[RT] looks for the row with the batch complete flag set to 0 to find the batch date for the current batch. It is a severe error for more than 1 row to have a batch complete flag set to 0. When the batch is complete the flag is set to 1. The first utility job (CTLBM01) that starts up each cycle checks that there is no row on the table with the batch complete flag set to 0 before inserting the new row. The last utility job that runs prior to completion (CTLBM02) updates the flag to 1 to indicate successful processing. |

Example DDL for the table is as follows:

```
create table dbo.ctl_batch_control
    (  pk_batch_number                          integer    not null
    ,  batch_date                               datetime    not null
    ,  batch_complete_flag                      integer    not null
    ,  constraint        pk_ctl_batch_control    primary key
       ( pk_batch_number
       )
    )
;
```

Example rows for this table are as follows:

| pk batch number | batch date | batch complete flag |
|---|---|---|
| 1 | 2006-01-02 | 1 |
| 2 | 2006-01-03 | 1 |
| 3 | 2006-01-04 | 0 |

The data in these rows has the following meaning.

Batch 1 complete successfully on 2006-01-02.
Batch 2 complete successfully on 2006-01-03.
Batch 3 has not completed successfully. It was started on 2006-01-04.

### 7.4.7. ctl_month_control

Some typical issues that often come up are:

- What dates are considered the first day of the month?
- What dates are used as default dates when you do not want to store nulls in date fields?

These issues are catered for with the ctl_month_control table.

The ctl_month_control table contains the following fields.

| Field Name | Description |
|---|---|
| pk_int_key | This is an integer primary key. There is only one record in this table so it can be any number you like. |
| last_month | This is the date that is considered to be the first day of the last month. |
| this_month | This is the date that is considered to be the first day of this month. |
| next_month | This is the date that is considered to be the first day of next month. |
| low_value_date | This is the date that is considered to be the default low date. |
| high_value_date | This is the date that is considered to be the default high date. |

Example DDL for the table is as follows:

```
create table dbo.ctl_month_control
    (  pk_int_key                          integer     not null
    ,  last_month                          datetime    not null
    ,  this_month                          datetime    not null
    ,  next_month                          datetime    not null
    ,  low_value_date                      datetime    not null
    ,  high_value_date                     datetime    not null
    ,  constraint     pk_ctl_month_control  primary key
       ( pk_int_key
       )
    )
;
```

An example row for this table is as follows:

| pk int key | last month | this month | next month | low value date | high value date |
|---|---|---|---|---|---|
| 1 | 2006-01-01 | 2006-02-01 | 2006-03-01 | 1900-01-01 | 9999-12-31 |

The data in this row has the following meaning.

The first day of last month is 2006-01-01, the first date of this month is 2006-02-01, the first day of next month is 2006-03-01. The default low value date is 1900-01-01. (Note that none of these fields are used in version 1.0. However they are there for future code already planned.)

The default high value date is 9999-12-31. This date is placed in the ‘date_to’ column of type 2 dimension records when they are inserted to say that the data is current and stretches out into the future.

### 7.4.8. ctl_codes_lookup

One of the things we come across again and again is the storage of lots of codes in a data warehouse. It has happened so often we have created a simple table to store them. This table is not used in SeETL[RT], however you might find this little piece of advice handy.

The ctl_codes_lookup table contains the following fields.

| Field Name | Description |
|---|---|
| pk_code_name | This is a name given to the code itself. For example it might be 'PolicyStatus' for the code table that records policy statuses. |
| pk_date_from | The date from which the code is valid. |
| pk_date_to | The date to which the code is valid. |
| pk_current_flag | A flag to say this is the current record. This table can record changes to meanings of codes over time. |
| code_value | The code value itself. For example for policy Statuses this might be '-1' meaning closed. |
| code_sdesc | A short description of what the code means. This is the value of the code that is likely to be printed on reports or put onto screens. |
| code_sdesc | A long description of what the code means. This is the value of the code that is likely to be looked up by people for a detailed note on what the code means. |

Example DDL for the table is as follows:

```
create table dbo.ctl_codes_lookup
    (  pk_code_name                          varchar (30)    not null
    ,  pk_date_from                          datetime        not null
    ,  pk_date_to                            datetime        not null
    ,  pk_current_flag                       integer         not null
    ,  code_value                            varchar (10)    not null
    ,  code_sdesc                            varchar (15)    not null
    ,  code_sdesc                            varchar (255)   not null
    ,  constraint     pk_ctl_codes_lookup    primary key
       ( pk_code_name
       , pk_date_from
       , pk_date_to
       , pk_current_flag
       )
    )
;
```

### 7.4.9. ctl_mapping_ss

The table ctl_mapping_ss is the table that the SeETL Spreadsheet is stored in to enable analysis and reports to be writted on the SeETL Spreadsheet. Reports may be written on the SeETL Spreadsheet information in any tools that can read this table. The values of the columns are defined in detail inside the spreadsheet.

This ability to write reports directly against the mapping spreadsheet that is known to be 100% consistent with the implemented ETL code is a HUGE breakthrough for ETL Architects. This is a 'world first'.

```
create table dbo.ctl_mapping_ss
    (  pk_spreadsheet_name          varchar     (255)           default 'unknown'
      ,pk_spreadsheet_version       varchar     (255)           default 'unknown'
      ,pk_mapping_row               integer                     default 0
      ,Use_flag_1                   varchar     (255)           default 'unknown'
      ,Row_Type                     varchar     (255)           default 'unknown'
      ,Source_Name                  varchar     (255)           default 'unknown'
      ,Source_Owner                 varchar     (255)           default 'unknown'
      ,Source_Table                 varchar     (255)           default 'unknown'
      ,Source_Table_Ldesc           varchar     (4000)          default 'unknown'
      ,Source_Column_Number         integer                     default 0
      ,Source_Column_Name           varchar     (255)           default 'unknown'
      ,Source_Column_Ldesc          varchar     (4000)          default 'unknown'
      ,Use_flag_2                   varchar     (255)           default 'unknown'
      ,Source_Col_Data_Type         varchar     (255)           default 'unknown'
      ,Source_Col_Length            varchar     (255)           default 'unknown'
      ,Source_Col_Precision         varchar     (255)           default 'unknown'
      ,Source_Col_Scale             varchar     (255)           default 'unknown'
      ,Source_Col_Null              varchar     (255)           default 'unknown'
      ,Source_Col_Required          varchar     (255)           default 'unknown'
      ,SeETL_Table_Type             varchar     (255)           default 'unknown'
      ,Target_Owner                 varchar     (255)           default 'unknown'
      ,SeETL_View                   varchar     (255)           default 'unknown'
      ,SeETL_Column_Number          integer                     default 0
      ,SeETL_View_Column_Name       varchar     (255)           default 'unknown'
      ,Target_Table                 varchar     (255)           default 'unknown'
      ,Target_Table_Exists_Ind      varchar     (255)           default 'unknown'
      ,Target_Column                varchar     (255)           default 'unknown'
      ,Target_Column_Exists_Ind     varchar     (255)           default 'unknown'
      ,Target_Column_Key_Ind        varchar     (255)           default 'unknown'
      ,Use_In_PV_Ind                varchar     (255)           default 'unknown'
      ,PV_Owner                     varchar     (255)           default 'unknown'
      ,PV_View                      varchar     (255)           default 'unknown'
      ,PV_Column                    varchar     (255)           default 'unknown'
      ,PV_Column_Number             integer                     default 0
      ,Action_Reqd                  varchar     (255)           default 'unknown'
      ,Notes1                       varchar     (4000)          default 'unknown'
      ,Notes2                       varchar     (4000)          default 'unknown'
      ,Notes3                       varchar     (4000)          default 'unknown'
      ,User_Col_01                  varchar     (255)           default 'unknown'
      ,User_Col_02                  varchar     (255)           default 'unknown'
      ,User_Col_03                  varchar     (255)           default 'unknown'
      ,User_Col_04                  varchar     (255)           default 'unknown'
      ,User_Col_05                  varchar     (255)           default 'unknown'
      ,User_Col_06                  varchar     (255)           default 'unknown'
      ,User_Col_07                  varchar     (255)           default 'unknown'
      ,User_Col_08                  varchar     (255)           default 'unknown'
      ,User_Col_09                  varchar     (255)           default 'unknown'
      ,User_Col_10                  varchar     (255)           default 'unknown'
)
;
```

### 7.4.10.ctl_file_cycle_control

The table ctl_file_cycle_control is not used by SeETL. However, it is made available to increase the level of auditability of an ETL Subsystem. We have used it in many projects in the past.

The idea of the file cycle control is to create a control table which records the number of times a file has been processed and how many rows were processed each time. It is an 'increase' or 'improvement' on the audit processing. The audit processing logs the number of rows processed by each program. The audit table is not meant to be particularly queryable as it is just a log. The ctl_file_cycle_control is meant to be queryable, joinable and used as a part of a well designed audit process.

It is also possible, and we have done this on many projects, to update the file cycle control table such that the rows that are sent into the data warehouse are tagged with the batch number and the file cycle number. It is then possible to tell which batch and which file cycle a row was inserted into or last updated in the data warehouse. This has proven very valuable over the years for tracking rows and detecting the introduction of errors and correcting them.

For example, if it is known that a specific calculation error happened to be introduced on a particular date when some fixes were applied it is possible to only apply corrections to rows updated after that particular date.

There is not code to update the file cycle control tables. It is expected that the user write some small sql snippets or stored procedures to insert rows into this table.

It is recommended that careful consideration be given to using file cycle numbers if your data warehouse needs a slightly better level of auditing than just the audit logs.

```
create table dbo.ctl_file_cycle_control
    (  pk_view_name                                 varchar (255)     not null
    ,  pk_file_cycle_number                         integer           not null
    ,  table_name                                   varchar (255)     not null
    ,  batch_date                                   datetime          not null
    ,  batch_number                                 integer           not null
    ,  number_of_records                            integer           not null
    ,  constraint          pk_ctl_file_cycle_control       primary key
       ( pk_view_name
       , pk_file_cycle_number
       )
    )
```

### 7.4.11.ctl_sequence_nums

The table ctl_sequence_nums table is used by SeETL to allocate sequence numbers to fact tables only.

The idea of sequence numbers on fact tables is this. There are many valuable and useful things that can be done by placing sequence numbers onto the front of fact tables. We have done many projects like this. However, SeETL did not originally support the generation of sequence numbers on fact tables. This support was left to the database. However, in some databases the introduction of sequence numbers at the database level is a significant performance issue. So we have added sequence numbers to SeETL. They are allocated during the attribution process and the allocation of the numbers is very fast.

The process is as follows.

If the user tells the attribution process to use sequence numbers and provides a sequence name then that name will be looked up by the attribution process to see what the last key allocated was. It will then update the last key allocated by seq_group_block size and it will give a sequence number to each row until it has exhausted all the sequence numbers in the seq_group_block. It will then return to this table to get another set of sequence numbers.

Currently this field is an integer so tables up to 4B rows are supported of negative sequence numbers are used. We will consider changing the sequence number to bigint when we have a client who has more than 4B rows in a single fact table using SeETL. That should be interesting!!

The seq_locked field has a specific purpose. In many cases a single set of input data to be processed is broken into multiple files. This is especially the case in telcos. However, we do not want sequence numbers to clash. In these cases the processing the multiple attribution processes will be going to the table to retrieve blocks of sequence numbers.

In order to make sure there is no overlap or duplication of sequence numbers each of the attribution processes can only retrieve keys when the seq_locked flag is 'N' before it starts the retrieval and update process.

Of course, this introduces contention on this table in the attribution processing. The Data Warehouse Architect must determine the most appropriate size of the seq_group_size such that:

1. Contention is minimised.
2. The gaps in the sequence numbers caused by unused numbers is not so large that the number ranges will run out of valid integers any time soon.

We expect that there will be no problems for sequence numbers for any table where the number of rows is less than 3 billion in a single table.

```
create table dbo.ctl_sequence_nums
    (  pk_int_key             integer           not null primary key
    ,  sequence_name          varchar     (255) not null default 'unknown'
    ,  last_key_allocated     integer           not null default 0
    ,  seq_group_size         integer           not null default 0
    ,  seq_locked             varchar     (1)   not null default 'N'
    )
```

## 7.5. DDL For Views

During the documentation thus far the term 'table' has been used interchangeably with the term 'view'. This is mainly because as far as SeETL[RT] is concerned everything is a view. It is strongly recommended that all tables passed to SeETL[RT] are views of tables, not the real tables themselves. This is to place no restrictions at all on your table names, only restrictions on view names.

### 7.5.1. Fact Table Processing

SeETL[RT] expects certain naming conventions to be used in order to understand how to take an input table and process it into a fact table.

These conventions are discussed in this section. We will continue to use the example of order details as the fact records.

#### *7.5.1.1. Input Table/View Naming Conventions*

The table (or view) which brings fact records into SeETL[RT] must have the following naming convention.

The character strings that will be used to perform lookups into the dimension tables must:

- Be at the front of the record.
- Must be named 'char_key_<dimension table name>
- Must be in the same order as the desired integer keys on the output fact record. This is one of the few instances where the order of the columns is important. This is because the column names are NOT the same on the input view and the target fact table.

As you can see from the input record the fields that are being used as the character keys are also passed through to the fact record. The 'char key' fields are discarded after the lookup process.

```
create table dbo.input_order_facts
    (
       char_key_time_dim             varchar    (0010)      not null
     , char_key_product_dim          varchar    (0006)      not null
     , char_key_customer_dim         varchar    (0006)      not null
     , char_key_vendor_dim           varchar    (0006)      not null
    ,  invoice_qty                   integer                not null
    ,  invoice_amt                   money                  not null
    ,  discount_amt                  money                  not null
    ,  product_code                  varchar    (0006)      not null
    ,  customer_code                 varchar    (0006)      not null
    ,  vendor_code                   varchar    (0006)      not null
    ,  invoice_num                   varchar    (0006)      not null
    ,  invoice_remark                varchar    (0030)      not null
    ,  order_date                    datetime               not null
    ,  payment_date                  datetime               not null
    ,  delivery_date                 datetime               not null
    )
;
```

#### *7.5.1.2. Dimension View Lookup Naming Conventions*

It is expected that you will use the actual dimension table name as the real table name within the data warehouse. Hence this name is not used by SeETL[RT]. In order to perform lookups you must use the following naming convention for your dimension table views.

- The view name must be called <dimension_table_name>_at. The 'at' is for 'attribution'.
- The concatenated string that is the unique key for the specific level in the dimension table must be called pk_dim_char_ky_field in the _at view.
- The primary key which is the integer key for the detailed level must be called 'dk_<dimension table name>
- The aggregate keys must be of the form <dimension table name>_key_ag<n> where n is the number of the aggregate.

From this set of rules the lookup processes can find the detailed level record based on the character string used to uniquely identify the dimension record at the detailed level. Only the detailed level need be looked up as it contains the keys for all higher levels.

An example for the customer_dim is provided below.

```
create view dbo.customer_dim_at
    (  pk_dim_char_ky_fld
    ,  dk_customer_dim
    ,  customer_dim_key_ag1
    ,  customer_dim_key_ag2
    ,  customer_dim_key_ag3
    ,  customer_dim_key_ag4
    ,  customer_dim_key_ag5
    ,  customer_dim_key_ag6
    ,  customer_dim_key_ag7
    ,  customer_dim_key_ag8
    ,  customer_dim_key_ag9
    )
as select
       dim_char_ky_fld
    ,  pk_customer_dim
    ,  customer_dim_key_ag1
    ,  customer_dim_key_ag2
    ,  customer_dim_key_ag3
    ,  customer_dim_key_ag4
    ,  customer_dim_key_ag5
    ,  customer_dim_key_ag6
    ,  customer_dim_key_ag7
    ,  customer_dim_key_ag8
    ,  customer_dim_key_ag9
    from dbo.customer_dim
    where level_col                    = 'detail'
```

#### *7.5.1.3. Detail Fact Table Naming Conventions*

The detailed level fact table must have the following naming conventions:

- The dimension table keys must be placed on the table/view in the same order as the keys sent in on the input record.
- The name of the dimension table keys must be pk_<dimension table name>.
- The primary keys must not be nullable. This is mandatory in most databases managers in any case.
- It is normal to put a unique index constraint on the concatenation of the primary keys to avoid accidental double processing or double inserting.

Again, this is actually expected to be a view. It is shown here as a table to provide the reader with a better idea of the data types of the columns.

```
create table dbo.f_order_fact
    (  pk_time_dim                  integer                 not null
    ,  pk_product_dim               integer                 not null
    ,  pk_customer_dim              integer                 not null
    ,  pk_vendor_dim                integer                 not null
    ,  invoice_qty                  integer                 not null
    ,  invoice_amt                  money                   not null
    ,  discount_amt                 money                   not null
    ,  product_code                 varchar        (0006)   not null
    ,  customer_code                varchar        (0006)   not null
    ,  vendor_code                  varchar        (0006)   not null
    ,  invoice_num                  varchar        (0006)   not null
    ,  invoice_remark               varchar        (0030)   not null
    ,  order_date                   datetime                not null
    ,  payment_date                 datetime                not null
    ,  delivery_date                datetime                not null
    )
;


create unique index f_order_factix1 on dwh.f_order_fact
    (  pk_time_dim
    ,  pk_product_dim
    ,  pk_customer_dim
    ,  pk_vendor_dim
    )
;
```

#### *7.5.1.4. Summary Fact Table Naming Conventions*

The summary fact table must have the following naming conventions:

- The view name must be <fact table name>_summary.
- The first column must be pk_aggregate_number.
- The dimension table keys must be placed on it in the same order as the detail fact table.
- The integer key columns must be called pk_<dimension table name>
- The data columns on the table must be able to be summed by the database manager.
- The primary keys must not be nullable.
- A unique index constraint on the concatenation of the primary keys is mandatory to provide support for incremental updates to the summary.

Again, this is actually expected to be a view, however showing it here as a table give you a better idea of the data types of the columns.

```
create table dbo.f_order_fact_summary
    (  pk_aggregate_number             integer                 not null
    ,  pk_time_dim                    integer                 not null
    ,  pk_product_dim                  integer                 not null
    ,  pk_customer_dim                 integer                 not null
    ,  pk_vendor_dim                   integer                 not null
    ,  invoice_qty                     integer                 not null
    ,  invoice_amt                     money                   not null
    ,  discount_amt                    money                   not null
    )
;


create unique index f_order_fact_summaryix1 on dbo.f_order_fact_summary
    (  pk_aggregate_number
    ,  pk_time_dim
    ,  pk_product_dim
    ,  pk_customer_dim
    ,  pk_vendor_dim
    )
;
```

#### *7.5.1.5. Sort Work Tables Naming Conventions*

It is unfortunate, but there is no standard sort package guaranteed to be available on any machine running C++. This is unlike cobol where there is a cobol sort for files supplied with the language. Given this restriction we decided against writing a sort program and using the database to perform sorts. (For those of you who want to use your sort package we would be willing to integrate it into the aggregation process on a fee basis.)

To perform sorts two sort work tables per fact table are required. They must have the following naming conventions:

- They must be called <fact table name>_swk1 and _swk2.
- They must be of exactly the same column definitions as the <fact table name>_summary view.
- The first column must be pk_aggregate_number.
- The integer key columns must be called pk_<dimension table name>
- The data columns on the table must be able to be summed by the database manager.

An example is provided below.

```
create table dbo.f_order_fact_swk1
    (  pk_aggregate_number            integer                 not null
    ,  pk_time_dim                    integer                 not null
    ,  pk_product_dim                 integer                 not null
    ,  pk_customer_dim                integer                 not null
    ,  pk_vendor_dim                  integer                 not null
    ,  invoice_qty                    integer                 not null
    ,  invoice_amt                    money                   not null
    ,  discount_amt                   money                   not null
    )
;
```

There are two sort work tables because rows are written into the first sort work table and then they are sorted and summarised into the second sort work table using an sql statement. They are then read from the second sort work table and written to CTLF002.

## 7.5.2. Dimension Table Processing

SeETL[RT] has two dimension maintenance programs. One for each of type 1 and type 2 dimension processing requirements. Each of these programs have slightly different naming convention requirements.

### *7.5.2.1. Type 1 Dimension Processing Naming Conventions*

In this example the underlying dimension table is as follows.

```
create table dbo.customer_dim
    (  pk_customer_dim                integer                 not null  primary key
    ,  level_col                      varchar        (0010)   not null
    ,  dim_char_ky_fld                varchar        (0030)   not null
    ,  cust_code                      varchar        (0006)   not null
    ,  cust_name                      varchar        (0030)   not null
    ,  cust_addr_1                    varchar        (0030)   not null
    ,  cust_addr_2                    varchar        (0030)   not null
    ,  cust_phone                     varchar        (0020)   not null
    ,  cust_birthday                  datetime                not null
    ,  cust_household                 integer                 not null
    ,  cust_geocode                   integer                 not null
    ,  cust_sex                       varchar        (0001)   not null
    ,  customer_dim_key_ag1           integer                 not null
    ,  customer_dim_key_ag2           integer                 not null
    ,  customer_dim_key_ag3           integer                 not null
    ,  customer_dim_key_ag4           integer                 not null
    ,  customer_dim_key_ag5           integer                 not null
    ,  customer_dim_key_ag6           integer                 not null
    ,  customer_dim_key_ag7           integer                 not null
    ,  customer_dim_key_ag8           integer                 not null
    ,  customer_dim_key_ag9           integer                 not null
    )
;
```

In order to apply type 1 dimension maintenance to this table the following naming conventions apply.

- The view name must be <dimension table name>_dm. The _dm stands for dimension maintenance.
- The 'level column' must be called 'pk_level_col'.
- The string field that defines the character key for this level of the dimension table must be called 'pk_dim_char_ky_fld'
- The detailed level integer key must be called 'dk_'<dimension table name>
- The aggregate keys if there are any must conform to the format <dimension table name>_key_ag<n> where n is the number of the aggregate 1-9.

An example view is documented below. The reasons for the change to the columns that are considered primary keys is that during maintenance the level column and the dim_char_ky_fld together make up the required primary key. This primary key is used to generate select, insert and update statements to be prepared and sent to the database to improve performance of the maintenance process. This is also one of the reasons why it is obvious that it is not possible to simply read the primary key information from the database itself.

```
create view dwh.customer_dim_dm
    (  pk_level_col
    ,  pk_dim_char_ky_fld
    ,  dk_customer_dim
    ,  cust_code
    ,  cust_name
    ,  cust_addr_1
    ,  cust_addr_2
    ,  cust_phone
    ,  cust_birthday
    ,  cust_household
    ,  cust_geocode
    ,  cust_sex
    ,  customer_dim_key_ag1
    ,  customer_dim_key_ag2
    ,  customer_dim_key_ag3
    ,  customer_dim_key_ag4
    ,  customer_dim_key_ag5
    ,  customer_dim_key_ag6
    ,  customer_dim_key_ag7
    ,  customer_dim_key_ag8
    ,  customer_dim_key_ag9
    )
as select
       level_col
    ,  dim_char_ky_fld
    ,  pk_customer_dim
    ,  cust_code
    ,  cust_name
    ,  cust_addr_1
    ,  cust_addr_2
    ,  cust_phone
    ,  cust_birthday
    ,  cust_household
    ,  cust_geocode
    ,  cust_sex
    ,  customer_dim_key_ag1
    ,  customer_dim_key_ag2
    ,  customer_dim_key_ag3
    ,  customer_dim_key_ag4
    ,  customer_dim_key_ag5
    ,  customer_dim_key_ag6
    ,  customer_dim_key_ag7
    ,  customer_dim_key_ag8
    ,  customer_dim_key_ag9
    from dwh.customer_dim
```

#### *7.5.2.2. Type 2 Dimension Processing Naming Conventions*

In this example the underlying dimension table is as follows.

```
create table dwh.customer_dim2
    (  pk_customer_dim2             integer                 not null  primary key
    ,  level_col                    varchar        (0010)   not null
    ,  dim_char_ky_fld              varchar        (0030)   not null
    ,  date_from                    datetime                not null
    ,  date_to                      datetime                not null
    ,  current_flag                 integer                 not null
    ,  cust_code                    varchar        (0006)   not null
    ,  cust_name                    varchar        (0030)   not null
    ,  cust_addr_1                  varchar        (0030)   not null
    ,  cust_addr_2                  varchar        (0030)   not null
    ,  cust_phone                   varchar        (0020)   not null
    ,  cust_birthday                datetime                not null
    ,  cust_household               integer                 not null
    ,  cust_geocode                 integer                 not null
    ,  cust_sex                     varchar        (0001)   not null
    ,  customer_dim2_key_ag1        integer                 not null
    ,  customer_dim2_key_ag2        integer                 not null
    ,  customer_dim2_key_ag3        integer                 not null
    ,  customer_dim2_key_ag4        integer                 not null
    ,  customer_dim2_key_ag5        integer                 not null
    ,  customer_dim2_key_ag6        integer                 not null
    ,  customer_dim2_key_ag7        integer                 not null
    ,  customer_dim2_key_ag8        integer                 not null
    ,  customer_dim2_key_ag9        integer                 not null
    )
    on CTLOR01FG01
;
```

To perform type 2 dimension maintenance actually requires 2 views over the table. We will deal with each view separately. The first view is the view that will receive inserts into the dimension table.

The following naming conventions apply to the first view.

- The view name must be <dimension table name>_dm1. The _dm1 meaning dimension maintenance view 1.
- The level column must be called pk_level_col.
- The dim_char_ky_fld must be called pk_dim_char_ky_fld
- The current flag must be called pk_current_flag
- The detailed level integer key must be called ‘dk_’<dimension table name>.
- The starting date of the record must be called date_from.
- The ending date of the record must be called date_to.
- The aggregate keys if there are any must conform to the format <dimension table name>_key_ag<n> where n is the number of the aggregate 1-9.

An example of creating a type 2 dimension maintenance view 1 is provided below.

```
create view dwh.customer_dim2_dm1
    (  pk_level_col
    ,  pk_dim_char_ky_fld
    ,  pk_current_flag
    ,  dk_customer_dim2
    ,  date_from
    ,  date_to
    ,  cust_code
    ,  cust_name
    ,  cust_addr_1
    ,  cust_addr_2
    ,  cust_phone
    ,  cust_birthday
    ,  cust_household
    ,  cust_geocode
    ,  cust_sex
    ,  customer_dim2_key_ag1
    ,  customer_dim2_key_ag2
    ,  customer_dim2_key_ag3
    ,  customer_dim2_key_ag4
    ,  customer_dim2_key_ag5
    ,  customer_dim2_key_ag6
    ,  customer_dim2_key_ag7
    ,  customer_dim2_key_ag8
    ,  customer_dim2_key_ag9
    )
as select
       level_col
    ,  dim_char_ky_fld
    ,  current_flag
    ,  pk_customer_dim2
    ,  date_from
    ,  date_to
    ,  cust_code
    ,  cust_name
    ,  cust_addr_1
    ,  cust_addr_2
    ,  cust_phone
    ,  cust_birthday
    ,  cust_household
    ,  cust_geocode
    ,  cust_sex
    ,  customer_dim2_key_ag1
    ,  customer_dim2_key_ag2
    ,  customer_dim2_key_ag3
    ,  customer_dim2_key_ag4
    ,  customer_dim2_key_ag5
    ,  customer_dim2_key_ag6
    ,  customer_dim2_key_ag7
    ,  customer_dim2_key_ag8
    ,  customer_dim2_key_ag9
    from dwh.customer_dim2
```

The following naming conventions apply to the second view.

- The view name must be <dimension table name>_dm2. The _dm2 meaning dimension maintenance view 2.
- The integer key for the record must be called pk_<dimension table name>.
- The current flag must be called current_flag
- The ending date of the record must be called date_to.

An example of creating a type 2 dimension maintenance view 2 is provided below.

```
create view dwh.customer_dim2_dm2
    (  pk_customer_dim2
    ,  current_flag
    ,  date_to
    )
as select
       pk_customer_dim2
    ,  current_flag
    ,  date_to

    from dwh.customer_dim2
```

This view is used to issue an update against the table to close out the existing record. Hence the need for the integer primary key to be exposed. Also, only the current flag and the date_to columns need to be updated. By using this view SeETL[RT] can update only those fields that need to be updated.

## 7.5.3. Association Table Processing

As SeETL[RT] developed it was clear that more functions could be easily added to it over time. Early in the process we took the decision to publish the functions that had proven so useful on so many projects:

- Managing dimension tables
- Attribution
- Aggregation
- Consolidation
- Loading

However, there are some functions that some people like to implement that SeETL[RT] did not cover. One of these are 'bridge' tables. Over the years, we have found bridge tables to be relatively limited in their applicability to large scale data warehousing.

Simply put, they create too many large sorts when running queries to be particularly useful. The classic example is analysing demographics of insurance customers by the various roles they play on an insurance policy. Though a bridge table seemingly provides the ability to perform such analysis, the processor costs of the sort through the policy to customer bridge table is too high to perform particularly interactive analysis on such a data structure. There are better ways. (Though we are not publishing those better ways here.)

### *7.5.3.1. Introduction to Associations*

However, there is an extension to a bridge table that is very useful. This is the 'association' table.

What is an association table? An association table is a table that associates two or more dimensions over a period of time.

The simplest example of an association table is the association of a customer with an address. We will use this example of an association in this document.

Customers can have many addresses. For example:

- Home address
- Office address
- PostBox address

Customers also change addresses on a regular basis. So addresses are quite time variant. One frequently used way of representing this has been a dimension table with a type 2 customer dimension table. Indeed, this is so normal the example type 2 dimension table supplied with SeETL[RT] is exactly like this. If, on the type 2 dimension table below cust_addr1/2 are tracked for changes a new dimension record would be written to this table when the address changed.

```
create table dbo.customer_dim2
    (  pk_customer_dim2         integer                 not null  primary key
    ,  level_col                varchar        (0010)   not null
    ,  dim_char_ky_fld          varchar        (0030)   not null
    ,  date_from                datetime                not null
    ,  date_to                  datetime                not null
    ,  current_flag             integer                 not null
    ,  cust_code                varchar        (0006)   not null
    ,  cust_name                varchar        (0030)   not null
    ,  cust_addr_1              varchar        (0030)   not null
    ,  cust_addr_2              varchar        (0030)   not null
    ,  cust_phone               varchar        (0020)   not null
    ,  cust_birthday            datetime                not null
    ,  cust_household           integer                 not null
    ,  cust_geocode             integer                 not null
    ,  cust_sex                 varchar        (0001)   not null
    ,  customer_dim2_key_ag1    integer                 not null
    ,  customer_dim2_key_ag2    integer                 not null
    ,  customer_dim2_key_ag3    integer                 not null
    ,  customer_dim2_key_ag4    integer                 not null
    ,  customer_dim2_key_ag5    integer                 not null
    ,  customer_dim2_key_ag6    integer                 not null
    ,  customer_dim2_key_ag7    integer                 not null
    ,  customer_dim2_key_ag8    integer                 not null
    ,  customer_dim2_key_ag9    integer                 not null
    )
```

This solution works and we have used it many times. But what are the 'unworkable' features of such a table? What don't we like about it?

1. There are many addresses and customers change addresses on a regular basis. Therefore this table becomes large as old addresses are closed out and new addresses open up. Analysis by this table therefore becomes more expensive as more addresses are added.

2. Addresses are unique, but this model treats the address as an attribute of customer. It is not. An address is something in it's own right. It is not an attribute of a customer, it is merely another 'thing' that is associated with the customer. For example, in a bank, a customer might actually move to address that was for a previous customer when buying a house. This model treats the address as an attribute of the customer.

#### *7.5.3.2. Association Tables – A Better Way*

The constant question we ask on our data warehouse projects is "Is there a better way?" With these types of association the answer is , yes.

The better way is to:
1. Make 'Address' a dimension in it's own right.
2. The 'Address' then is stored in a dimension table which might be type 1 or type 2 as you wish. This implies there is an Address_Key on the address_dim table.
3. Associate the customer with the address via an association table.

Note, that you can also put the current address_key onto any fact table at any time to join the address directly to the fact table thus removing the need to go via the association table to get to the address for all queries. This provides a massive increase in speed for address based analysis.

#### *7.5.3.3. What is an Association Table?*

The association table then has the following fields on it:

- Association Table Primary Key
- Dimension_Table_1_Key
- Dimension_Table_2_Key
- Date_From
- Date_To
- Current_Flag
- Association_Reason (If you want it. It is not mandatory)
- Dimension_Table_1_dim_char_key_fld (used to look up the 2 keys)

The example customer to address association table is as follows:

```
create table dbo.customer_address_asoc
    (  pk_customer_address_asoc        integer                  not null primary key
    ,  dk_customer_dim                 integer                  not null
    ,  dk_address_dim                  integer                  not null
    ,  dk_date_from                    integer                  not null
    ,  dk_date_to                      integer                  not null
    ,  date_from                       datetime                 not null
    ,  date_to                         datetime                 not null
    ,  current_flag                    integer                  not null
    ,  asoc_key_customer_address_asoc  varchar        (0255)    not null
    ,  cust_code                       varchar        (0006)    not null
    ,  address_code                    varchar        (0255)    not null
    ,  address_role                    varchar        (0255)    not null
   )
```

As you can see from the example. This table will associate the customer and the customers address for a variety of 'address roles'. The cust_code and address_code (the real keys) have also been placed on the record redundantly.

#### *7.5.3.4. Association Table Naming Conventions*

The underlying association table can have any column names. However, the view of the table presented to SeETL[RT] must have the following naming conventions:

- The primary key must be the first column and it must be called pk_<association table name>.
- The name of the dimension table keys must be dk_<dimension table name>. They must be presented in the same order as they are presented on the input table.
- The date from and date to keys must be called dk_date_from and dk_date_to.
- The actual date from and date to must be called date_from and date_to.
- The current flag must be called current_flag.
- The character string that will be used as the unique key to determine if the association exists must be called asoc_key_<association table name>.
- The primary key must not be nullable. This is mandatory in most databases managers in any case.
- The detail keys should not be nullable.
- Any other columns can be placed at the end of the table. There is no limit inside SeETL[RT] for how many columns can be placed at the end of the table.
- It is recommended that the real keys are also placed on the end of the table so that it is easier to see that the association has worked correctly. This is optional.
- It is normal to put a unique index constraint on the primary key.

#### *7.5.3.5. Association Table Input Table Naming Conventions*

The input table for an association table must have the following naming conventions:

- The character keys that will be used to look up dimensions must be called char_key<dimension table name>. This is exactly like a dimension table naming standard.
- The character string that will be used as the key from the input record must be called asoc_key_<association table name>.
- Other fields can be named any valid column name for your database and ODBC.

An example input table for an association table is as follows:

```
create table dbo.in_customer_address
    (  char_key_customer_dim           varchar        (0255)    not null
    ,  char_key_address_dim            varchar        (0255)    not null
    ,  asoc_key_customer_address_asoc  varchar        (0255)    not null
    ,  cust_code                       varchar        (0255)    not null
    ,  address_role                    varchar        (0255)    not null
    ,  address_code                    varchar        (0255)    not null
    ,  address_line_1                  varchar        (0255)    not null
    ,  address_line_2                  varchar        (0255)    not null
    ,  address_line_3                  varchar        (0255)    not null
    )
```

#### *7.5.3.6. Association Tables View Naming Conventions*

Obviously, the input table does not contain the date keys. So the question is where do they come from? An association table runs on a 'batch processing date' which is stored in the ctl_batch_control table. The closing date of an association table is assumed to be 'batch processing date – 1 day'. To retrieve the batch processing date – 1 day the following view must be defined and available to the association program. Note that dim_date is a date or datetime datatype and dk_time_dim is the integer key for that date record.

```
create view dbo.time_dim_asoc_lkp
    (  pk_time_dim
    ,  dk_time_dim
    )
as select
       dim_date
    ,  pk_time_dim
    from dbo.time_dim
    where level_col                    = 'detail'
;
```

The process is that the association program reads the batch processing date from the ctl_batch_control table and then it performs two lookups to the time_dim_prfl_lkp table one for batch processing date – 1 day and one for batch processing date. In this way the association program has available to it:

- Batch Processing Date
- Batch Processing Date – 1 day
- Integer keys for both

Associations are valid for periods of time. People live at addresses for more than one day. So it is necessary to detect changes in associations. For example, if a customer moves address the operational system will send a record into SeETL[RT] with the new customer/address association information on it in the dimension key fields. SeETL[RT] must detect this change. Also, to make it easier for the operational systems, SeETL[RT] must also be able to detect the fact that no change happened and discard the input record.

It does this through a number of views.

**Association_table_1_lkp1**

This view is required to be able to determine if this particular set of associations currently exist. The way that it is used is that the input record is 'attributed' just like a fact record and then the association program performs a lookup on all the keys derived in the attribution process to find the integer key of the association record. If the record is found the program knows that the association already exists and the input record can be discarded. If the record is not found then the program knows it must perform some processing. And the next view determines what processing. The example lookup table is included below.

```
create view dbo.customer_address_asoc_lkp1
    (   pk_customer_dim
    ,   pk_address_dim
    ,   pk_current_flag
    ,   dk_customer_address_asoc
    )
as select
        dk_customer_dim
    ,   dk_address_dim
    ,   current_flag
    ,   pk_customer_address_asoc
    from dbo.customer_address_asoc
    where current_flag = 1
    ;
```

**Association_table_1_lkp2**

When no record is found in association_table_1_lkp1 there are two conditions that might be true:

1. The association may be new. For example a new customer. A new customer will not have any association record for an address. In this case, the association record must be inserted with the batch processing date as the date from and the high date as the date to.

2. The association may have changed. For example, the customer may have moved address. The customer had a previous address and we knew about the previous address. In this case the association program must close the previous association. The date to of the previous association is recorded as the batch processing date – 1 day. The insert of the new association is exactly the same as the insert for the association not having previously existed.

To detect whether the association existed previously we must lookup the association record. An example view to do this is as follows:

```
create view dbo.customer_address_asoc_lkp2
    (   pk_asoc_key_customer_address_asoc
    ,   pk_current_flag
    ,   dk_customer_address_asoc
    )
as select
        asoc_key_customer_address_asoc
    ,   current_flag
    ,   pk_customer_address_asoc
    from dbo.customer_address_asoc
    ;
```

This is where it get's a little tricky. Obviously if you make the asoc_key for association table 1 the dim char key fld of both dimension tables and the reason for the association the program cannot detect the old record. It is much more likely that you will make the asoc_key_ the dim_char_ky_fld from dimension table 1 concatenated with the reason for the association. This way when a person changes home address the program will see the customer key and the role of 'home' and detect that a current home address association exists.

**Association_table_1_upd**

Should the program detect that a current association exists then it must close the current association record prior to inserting the new record. To minimise the number of fields updated another view is required to apply the update. That view is as follows:

```
create view dbo.customer_address_asoc_upd
    (   pk_customer_address_asoc
    ,   dk_date_to
    ,   date_to
    ,   current_flag
    )
as select
        pk_customer_address_asoc
    ,   dk_date_to
    ,   date_to
    ,   current_flag
    from dbo.customer_address_asoc
    ;
```

The association program will use the dk_customer_address_asoc integer retrieved from dbo.customer_address_asoc_lkp2 as the key to dbo.customer_address_asoc_upd to close out the previous record.

**Association_table_1_ins**

Because you may use the name ‘association_table’ as the real table name inserts to the association table are via a view called <association_table_name>_ins.

The example insert view is as follows:

```
create view dbo.customer_address_asoc_ins
    (   pk_customer_address_asoc
    ,   pk_customer_dim
    ,   pk_address_dim
    ,   dk_date_from
    ,   dk_date_to
    ,   date_from
    ,   date_to
    ,   current_flag
    ,   asoc_key_customer_address_asoc
    ,   cust_code
    ,   address_code
    ,   address_role

    )
as select
        pk_customer_address_asoc
    ,   dk_customer_dim
    ,   dk_address_dim
    ,   dk_date_from
    ,   dk_date_to
    ,   date_from
    ,   date_to
    ,   current_flag
    ,   asoc_key_customer_address_asoc
    ,   cust_code
    ,   address_code
    ,   address_role
    from dbo.customer_address_asoc
```

#### *7.5.3.7. Other Interesting Aspects of Association Tables*

**Can an association be between only 2 dimension tables?**

No. We decided to write the code so that you could associate as many different dimensions together over a period of time. For example, you may want to also record email addresses and telephone numbers. These can be attributes of the customer or they can be dimension tables in their own right and linked to the same association table using Dimension_Table_3_Key and Dimension_Table_4_Key. The number of things that can be associated are only limited by the number of columns on a table in the database.

**Where Does the Primary Key Come From?**

As an added feature association tables include an integer primary key as the first field. This is very useful for the lookups and updates. The 'last key used' is retained in the ctl_last_key_used table just like the 'last key used' for dimension tables. Thus, you must include an entry in the ctl_last_key_used table for the association table prior to running the association table.

**Error Message CTLS0203**

The most common error message received during testing of this program is CTLS0203 with an ODBC error of 'Invalid character for cast specification'. Unfortunately, this is the Microsoft ODBC error message when the character representation of a data type cannot be converted to the database type for the column. In the association program this is almost always due to the char_key<dimension_table_name> on the input record not reconciling with the pk_<dimension_table_name> on the <association_table>_lkp1 table.

In testing we initially tried to initialise the keys to be zero prior to the lookup into the lkp1 table. However, this meant that the lookup 'appeared' to work and a zero key was returned. We felt this was a harder to find error than the termination of the program so we reverted back to terminating the program under such conditions. Hence, if you receive error message CTLS203 along with the 'Invalid character for cast specification' or it's equivalent on other databases check the char_key<dimension_table_name> on the input record with the pk_<dimension_table_name> on the <association_table>_lkp1 table. This is almost certainly your problem.

**The Association Table Looks Like it Can Be a Snowflake**

If you notice that an association table looks like it could act as snowflake then you are absolutely correct. Indeed, we have publicly stated on many occasions our disagreement with 'snowflaking' for the sake of 'snowflaking'. However, there are some cases when building a hierarchy in dimension tables is useful. In these cases, we recommend that integer keys be used to link the hierarchy of dimension tables together, and we usually suggest that the links be time variant. So in this case an association table can also act like a type 2 dimension table with three keys on it. One for the association dimension record itself, one for the child dimension record and one for the parent dimension record.

Note that if you want to build a hierarchy of dimensions using integer keys and linking type 2 dimension tables you must also maintain some 'working' type 2 dimension tables that contain just the keys, time variant and dim_char_ky_fld portions of the dimension table. These 'working' dimension tables must be updated prior to the snowflake dimension tables. This is because to place the integer keys for both the parent and child dimensions on to a dimension table both the parent and child dimension rows must exist. This is not possible for three or more dimensions in a hierarchy because the parent and child tables cannot all exist for all dimensions for new rows.

Such an association table will keep the people who like to snowflake happy.

#### *7.5.3.8. Running an Association Table Processing Program*

When you run an instance of the association table program it does not actually perform the inserts and updates of the association table. It writes out files which can then be applied by the Data Transfer Utility. This is to ensure that the association program is re-runnable from the beginning in the event of some system failure.

The command to run the association program is as follows:

```
CTLAS01.exe DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=sa;PWD=password;DATABASE= SEETL3000
InCatalogName= SEETL3000 InSchemaName=dbo  InTableName=in_customer_address
OutTableName=customer_address_asoc CTLF007FileName=D:\
IBISoftware\SeETL\DesignTime\3.0.00\TestData\CTLF007.DAT CTLF008FileName=D:\
IBISoftware\SeETL\DesignTime\3.0.00\TestData\CTLF008.DAT ErrorMessageOutput=cerr DebugLevel=9 Audit=No
```

You should then create two more command files. One to insert the records from CTLF007 and one to apply the CTLF008 records as updates.

The command to insert rows is as follows. Notice that the insert table name ends in '_INS'. This is required. Also notice that the InsertUpdateOption is set to InsertOnly because this file should only contain inserts. Any primary key clash would be an error.

```
CTLU001.exe DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=sa;PWD=password;DATABASE=SEETL3000
OutCatalogName=SEETL3000 OutSchemaName=dbo  OutTableName=customer_address_asoc_ins
InsertUpdateOption=InsertOnly DataMovementOption=Load WorkFileName=D:\
IBISoftware\SeETL\DesignTime\3.0.00\TestData\CTLF007.DAT LoadFactTable=No  Audit=Yes DebugLevel=9
ErrorMessageOutput=cerr
```

The command to update rows is as follows. Notice that the update table name ends in '_UPD'. This is required. Also notice that the InsertUpdateOption is set to UpdateThenInsert because this file should only contain updates. Thus it saves some processing time to issue updates first.

```
CTLU001.exe DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=sa;PWD=password;DATABASE=SEETL3000
OutCatalogName=SEETL3000 OutSchemaName=dbo  OutTableName=customer_address_asoc_upd
InsertUpdateOption=UpdateThenInsert DataMovementOption=Load WorkFileName=D:\
IBISoftware\SeETL\DesignTime\3.0.00\TestData\CTLF008.DAT LoadFactTable=No  Audit=Yes DebugLevel=9
ErrorMessageOutput=cerr
```

#### *7.5.3.9. Example Data From An Association Table*

To provide an example of the data that might be stored in an association table consider the two tables below. The first table provides example input to the association process.

Assume there is an 'address_dim' table which has been populated with all the different addresses and the address code is the only field used as the char key into the address_dim.

The input table has the char key for the customer dimension and the char key for the address. It also contains a string field which is the customer char key and the address role. This makes it possible to determine if we have a particular type of address for a particular customer during the update process. The sample table then contains the real values of customer code, address role and address code. It also contains address lines which are not shown in the table below.

| Char Key Customer Dim | Char Key Address Dim | Asoc Key Customer Address Asoc | Cust Code | Address Role | Address Code |
|---|---|---|---|---|---|
| CU0001 | AD0004 | CU0001Home | CU0001 | Home | AD0004 |
| CU0001 | AD0002 | CU0001Work | CU0001 | Work | AD0002 |
| CU0001 | AD0003 | CU0001PostBox | CU0001 | PostBox | AD0003 |
| CU0002 | AD0001 | CU0002Home | CU0002 | Home | AD0001 |
| CU0003 | AD0004 | CU0003Home | CU0003 | Home | AD0004 |

Having processed this data through the association program the following output is generated. Please note that during the test case Customer CU0001 'moved' from address AD0001 to AD0004 as his/her 'Home' Address. Hence in the table below you can see a closed record for CU001Home and another open record for CU001Home.

Clearly the test cases show that the customer and the address are being associated via the association table.

| Pk Customer Address Asoc | Dk Customer Dim | Dk Address Dim | Dk Date From | Dk Date To | Date From | Date To | Current Flag | Asoc Key Customer Address Asoc |
|---|---|---|---|---|---|---|---|---|
| 29 | 23 | 10 | 0 | 0 | 5/12/2000 | 7/12/2000 | 0 | CU0001Home |
| 30 | 23 | 11 | 0 | 9999999 | 5/12/2000 | 31/12/9999 | 1 | CU0001Work |
| 31 | 23 | 12 | 0 | 9999999 | 5/12/2000 | 31/12/9999 | 1 | CU0001PostBox |
| 32 | 27 | 10 | 0 | 9999999 | 5/12/2000 | 31/12/9999 | 1 | CU0002Home |
| 33 | 31 | 13 | 0 | 9999999 | 5/12/2000 | 31/12/9999 | 1 | CU0003Home |
| 34 | 23 | 13 | 0 | 9999999 | 8/12/2000 | 31/12/9999 | 1 | CU0001Home |

Association tables play a vital role in more complex data warehouse projects.

# 8. PARAMETERS

This section documents the valid parameters for all programs. In general parameters that are provided to a program that are not needed are ignored. Parameters that are required by a program and not provided produce an error condition and stop processing.

## 8.1. Definition of Parameters

| Parameter Name | Description and valid values |
|---|---|
| DBConnectionInParameter | This is the ODBC DSN Connection information that is required to issue the connection to the source ODBC DSN. This varies from database manager to database manager and you should consult your particular database provider for details of the contents of this parameter.<br><br>For SQL Server 2000 a valid connection parameter is as follows:<br>DSN=SEETL3000;SERVER=PETERLAP;UID=sa;<br>PWD=password;DATABASE=SEETL3000;<br><br>For Oracle 9 a valid connection parameter is as follows:<br>DSN=ORASEETL3000;SERVER=ORAGDB1.PETERLAP;UID=system;<br>PWD=manager;<br><br>For Access 2000 a valid connection parameter is as follows:<br>DSN=ACCSEETL3000;SERVER=PETERLAP;UID=admin;PWD=;<br>DATABASE= ACCSEETL3000;<br><br>Default Value:<br>“Not Set” |
| InCatalogName | This is the ‘Catalog’ name that forms the first part of the fully qualified table name from which data will be read. For most databases this is the database name itself. For example if connecting with the above SQL Server 2000 connection string the CatalogName would be ‘SEETL3000’. This parameter is ignored for MS Access.<br><br>Default Value:<br>“Not Set” |
| InSchemaName | This is typically the ‘owner’ of the table. It forms the second component of the fully qualified table name from which data will be read. This parameter is ignored for MS Access.<br><br>Default Value:<br>“Not Set” |
| InTableName | This is the table/view name that forms the third component of the fully qualified table name from which data will be read.<br><br>Default Value:<br>“Not Set” |
| DBConnectionOutParameter | This is the ODBC DSN Connection information that is required to issue the connection to the target ODBC DSN. By using different connection strings for source and target you can move data from one database to another.<br>If the DBConnectionOutParameter is not set and the DBConnectionInParameter is set the DBConnectionOutParameter defaults to the DBConnectionInParameter.<br><br>Default Value:<br>“Not Set” |
| OutCatalogName | This is the ‘Catalog’ name that forms the first part of the fully qualified table name to which data will be written. If OutCatalogName is not set and InCatalogName is set then OutCatalogName will default to InCatalogName. This parameter is ignored for MS Access. |

| | |
|---|---|
| | Default Value:<br>“Not Set” |
| OutSchemaName | This is typically the ‘owner’ of the table. It forms the second component of the fully qualified table name to which data will be written. If OutSchemaName is not set and InSchemaName is set then OutSchemaName will default to InSchemaName. This parameter is ignored for MS Access.<br><br>Default Value:<br>“Not Set” |
| OutTableName | This is the table/view name that forms the third component of the fully qualified table name to which data will be written. If OutTableName is not set and InTableName is set then OutTableName will default to InTableName.<br><br>Default Value:<br>“Not Set” |
| DBConnectionMsgParameter | This is the ODBC DSN Connection information that is required to issue the connection to the message and audit ODBC DSN. By using a different connection string audit and message processing it is possible to centralise the storage of messages for audit and errors for all processing.<br>If the DBConnectionMsgParameter is not set and the DBConnectionOutParameter is set the DBConnectionMsgParameter defaults to the DBConnectionOutParameter.<br><br>Default Value:<br>“Not Set” |
| MsgCatalogName | This is the ‘Catalog’ name that forms the first part of the fully qualified table name to which messages will be written. If MsgCatalogName is not set and OutCatalogName is set then MsgCatalogName will default to OutCatalogName. This parameter is ignored for MS Access.<br><br>Default Value:<br>“Not Set” |
| MsgSchemaName | This is typically the ‘owner’ of the table. It forms the second component of the fully qualified table name to which messages will be written. If MsgSchemaName is not set and OutSchemaName is set then MsgSchemaName will default to OutSchemaName. This parameter is ignored for MS Access.<br><br>Default Value:<br>“Not Set” |
| WorkFileName | This is the fully qualified name of a file that the server has access to that can be opened for reading and writing. Data is unloaded to the work file and then loaded into the target database. An open (read or write) is issued to this name. An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlf001.dat<br>When loading data produced by CTLAT01/CL01 the parameter used to pass the file name to the program is this parameter. The file name to pass in is one of the file names that were used in the file name parameters below.<br><br>Default Value:<br>“Not Set” |
| KillFileName | This is the fully qualified name of a file that the server has access to that can be opened for reading and can be deleted by the processes that run. Some processes such as the Batch Processing Scheduling Utility and the Load Dimension Tables into Memory Maps Utility run ‘endlessly’ until told to stop somehow by the user. The way these ‘endless’ routines are told to stop is by using a ‘kill’ file. The user passes the KillFileName as a parameter, and then creates the file inside the operating system when he/she wants the process to stop.<br><br>An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlu0008.kill.yes |

| | |
|---|---|
| | Default Value:<br>“Not Set” |
| WaitFileName | The Load Dimension Tables into Memory Maps Utility runs endlessly and it is ended by the KillFileName parameter being found in the operating system. However, when using memory mapped IO the attribution process is dependent on the Load Dimension Tables into Memory Maps Utility. Since the Load Dimension Tables into Memory Maps Utility is ‘endless’ another mechanism is required to provide the dependency for the attribution processes AT01/02. This dependency is created by the WaitFileName. The Load Dimension Tables into Memory Maps Utility creates the WaitFileName parameter in the operating system when all dimension tables have been loaded into memory mapped files and they are ready for use by the attribution processing. Hence, when using memory mapped files for attribution processing the user should create a dependency on WaitFileName for the attribution processing.<br><br>An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlu012.wait.yes<br><br>Default Value:<br>“Not Set” |
| CTLF001FileName | This is the fully qualified name of a file that the server has access to that can be opened for reading and writing. This is the file that CTLAT01 writes attributed detail fact records to. It is only required for CTLAT01/CTLAG01. An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlf001.dat<br><br>This file name is also used as input to the Delimiter Separated Values Reformat Utility.<br><br>Default Value:<br>“Not Set” |
| CTLF002FileName | This is the fully qualified name of a file that the server has access to that can be opened for reading and writing. This is the file that CTLAG01 writes summarised fact records to. It is only required for CTLAG01/CL01. An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlf002.dat<br><br>This file name is also used as output from the Delimiter Separated Values Reformat Utility.<br><br>Default Value:<br>“Not Set” |
| CTLF003FileName | This is the fully qualified name of a file that the server has access to that can be opened for reading and writing. This is the file that CTLCL01 writes summary records which are to be inserted into the summary fact table to. It is only required for CTLCL01. An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data \ctlf003.dat<br><br>Default Value:<br>“Not Set” |
| CTLF004FileName | This is the fully qualified name of a file that the server has access to that can be opened for reading and writing. This is the file that CTLCL01 writes summary records which are to be applied to the summary fact table as updates. It is only required for CTLCL01. An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlf004.dat<br><br>Default Value:<br>“Not Set” |
| CTLF005FileName | This file name is reserved for processing specifically required for Sybase Industry Warehouse Studio.<br><br>Default Value: |

| | |
|---|---|
| | "Not Set" |
| CTLF006FileName | This file name is reserved for processing specifically required for Sybase Industry Warehouse Studio.<br><br>Default Value:<br>"Not Set" |
| CTLF007FileName | This is the fully qualified name of a file that the server has access to that can be opened for writing. This is the file that CTLAS01 writes detailed association records to when performing it's association processing.. An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlf007.dat<br>This file is also used by CTLAT02 which is an attribution program which has been specifically built for Sybase Industry Warehouse Studio.<br>Default Value:<br>"Not Set" |
| CTLF008FileName | This is the fully qualified name of a file that the server has access to that can be opened for writing. This is the file that CTLAS01 writes details of association records to close when performing it's association processing.<br><br>An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlf008.dat<br><br>Default Value:<br>"Not Set" |
| SQLFileName | This is the fully qualified name of a file that contains valid SQL which you want to process against a data source using the CTLU002 – SQL Batch Processor utility. The server must have read access to this file. It is only required for CTLU002.<br><br>An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\sqlcommand.dat<br><br>Default Value:<br>"Not Set" |
| InsertUpdateOption | The insert update option indicates the behaviour of the insert/update process. The valid values and their meanings are as follows:<br>• InsertOnly (Default Value)<br>Records are inserted into the target table. A unique index constraint violation is considered to be an error and processing will stop if a unique index constrain violation occurs.<br>• InsertThenUpdate<br>Records are inserted into the target table. If a unique index constraint violation occurs an update will be issued using the primary key of the table in the where clause for the update.<br>• InsertThenDeleteInsert<br>This function has been specifically added for Sybase IQ. The records are inserted into the target table. If a unique index constraint violation occurs a delete will be issued using the primary key of the table in the where clause for the delete. If the delete is successful then an insert will be issued.<br>• UpdateThenInsert<br>If the majority of rows are going to exist in the target database performing inserts first is a bit wasteful of processing time for large volumes. Update then insert allows you to issue an update first and then if the record is not found issue the insert.<br>This option is only valid for CTLU001.<br><br>Default Value:<br>InsertOnly |
| DataMovementOption | This defines how the data is to be moved in this particular run of the program. The valid values and their meanings are as follows:<br>• Transfer (Default Value)<br>Records are copied from the source table to the target table in the one invocation of the program. The work file is still used to contain data during the transfer. |

| | |
|---|---|
| | • Unload<br>Records are unloaded from the 'In' Catalog,Schema.Table to the workfile.<br>• Load<br>Records are loaded from the work file to the 'Out' Catalog.Schema.Table.<br>This option is only required for CTLU001.<br>Default Value:<br>Transfer |
| CreateLoadImageFile | The Data Transfer Utility can produce a 'Load Image File' which is simply a file that is a delimited file that can be provided to the database loader in order to load the file. This is an alternative to performing inserts/'updates to assist in loading large volumes of data.<br><br>If the parameter CreateLoadImage is set to 'Yes' then the parameter DataMovementOption is ignored because no data will actually be 'moved' by the Data Transfer Utility.<br><br>The attribution processing can also produce a Load Image File and AT01/02 accept the CreateLoadImageFile parameter.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value :<br>'No'. |
| LoadImageFileName | This is the fully qualified name of a file that the server has access to that can be opened for writing. Data is unloaded to the Load Image File ready to be loaded into a table using the native database load utility rather than the ODBC interface presented by the Data Transfer Utility.<br><br>The attribution processing can also produce a Load Image File and AT01/02 accept the LoadImageFileName parameter.<br><br>An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlf001.lif.<br><br>Default Value:<br>"Not Set" |
| DeleteRowToBeLoaded | The Data Transfer Utility can optionally delete the rows to be loaded from the Load Image File into the target table. This is to ensure no key/index clashes when the Load Image File is loaded into the database.<br><br>Most database loaders will not allow duplicate rows to be loaded by the loader and usually terminate such load processes. By deleting the row to be loaded the Data Transfer Utility can guarantee that the loader will load all rows properly and as quickly as possible. This option is specifically for Sybase IQ but has been built to support other databases as well.<br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'No'. |
| CreateHeaderRecord | The Data Transfer Utility can send data to downstream systems (such as database loaders) in a flat file format. The DTU can optionally create a header row containing column names. If the user does NOT want a header record with column names in the output file this option can be set to 'No'.<br>Valid values are:<br>• Yes<br>• No<br><br>Default Value: |

| | |
|---|---|
| | 'Yes'. |
| UseNullCharacter | When sending data to downstream systems (such as database loaders) using the Data Transfer Utility, especially database loaders, a character must be specified to indicate that the field is null. Using the null character is optional. This parameter specifies if a null character is to be placed into the output file to the database loaders.<br><br>There is also a need to be able to detect nulls in incoming data. For example, a fixed format file or a delimited file may have a character specified by the source system that represents the NULL character. In this case this special character can be translated into a NULL in the file reformat utilities.<br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'Yes'. |
| NullCharacter | When sending data to downstream systems (such as database loaders) using the Data Transfer Utility, especially database loaders, a character must be specified to indicate that the field is null. This character is specified in the load script for the table and is interpreted by the loader as null. SeETL[RT] will take the NullCharacter specified and produce a field of length 1 with this character in it. Where this is the case the loading script must interpret the field as a null.<br><br>There is also a need to be able to detect nulls in incoming data. For example, a fixed format file or a delimited file may have a character specified by the source system that represents the NULL character. In this case this special character can be translated into a NULL in the file reformat utilities.<br><br>Note. In SeETL 3.0.00 we have introduced a special case for detecting nulls in a delimited file. If a file is delimited or fixed format and you would like any field that is not present in the file to be converted to a NULL then this parameter should be set to "NULL". When this parameter is set to NULL then SeETL will set any field that is found to be a zero length character string to NULL.<br><br>This is particularly useful in those cases when a source system is sending numeric fields in files but it does not provide a null character when no value is present it just sends blanks. In these cases it is now possible to accept blank values for numeric fields by defaulting them to NULL. Indeed, it is possible to accept blank characters for any type of field by defaulting them to NULL.<br><br>Valid Values: Any ascii character.<br><br>Default Value:<br>'^'. |
| UseQuoteCharacter | When sending data to downstream systems (such as database loaders) using the Data Transfer Utility, especially database loaders, a character may be specified for the quote character. Using the quote character is optional. This parameter specifies if a quote character is to be placed into the output file around each of the fields that require quotes. The fields that are currently defined to require quote characters are as follows:<br>• SQL_CHAR<br>• SQL_VARCHAR<br>• SQL_LONGVARCHAR<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'Yes'. |

| | |
|---|---|
| QuoteCharacter | When sending data to downstream systems (such as database loaders) using the Data Transfer Utility, especially database loaders, a character may be specified as the quote character. This character is specified in the load script for the table and is interpreted by the loader as a quote. SeETL[RT] will use this QuoteCharacter when writing the LoadInterfaceFile. Note that because the quote character terminates the parameter string passed to any given program the escape character must be placed before it to pass the quote character properly. Eg. \" or \'.<br>Valid Values: Any ascii character.<br><br>Default Value:<br>". |
| LIFDelimiterCharacter | The Data Transfer Utility can produce a 'Load Image File' which is simply a file that is a delimited file that can be provided to the database loader in order to load the file. This is an alternative to performing inserts/'updates to assist in loading large volumes of data.<br><br>The user may specify the delimiter to use in the Load mage File in the LIFDelimiterCharacter parameter.<br>Note that no validation will be performed on the delimiter passed to the Data Transfer Utility. If the value passed is longer then one character it will be ignored.<br><br>Valid Values are:<br>Any single ascii character.<br><br>Default Value:<br>',' |
| LoadFactTable | This defines whether the table being loaded is a fact table produced by SeETL[RT]. The default value is 'No' and the only valid value for the parameter is 'Yes'. The purpose of this option is to search through the column names in the work file looking for columns with a prefix of 'dk_' or 'DK_'. If it finds any column with such a prefix it changes the prefix to 'pk_' or 'PK_' to send it into the fact table. This option can be ignored unless you are loading fact tables.<br>This option is only required for CTLU001.<br><br>Default Value:<br>"No" |
| SourceDatabase | This defines the type of database from which data will be read. It is required because there are some differences in the ODBC statements that can be processed by each database. The valid values and their meanings are as follows:<br>• SQLSERVER2000 (Default Value)<br>This specifies that the source database is SQL Server 2000.<br>• SQLSERVER2005<br>This specifies that the source database is SQL Server 2005.<br>• MSACCESS<br>This specifies that the source database is MS Access. Currently only MS Access 2000 has been tested though it is expected that Access 2002 will also work. Note that MS Access is supported for the Data Transfer Utility and as a source for SeETL[RT]. MS Access is not supported as a target database for SeETL[RT].<br>• ORACLE8<br>This specifies that the source database is Oracle 8.<br>• ORACLE9<br>This specifies that the source database is Oracle 9.<br>• ORACLE10<br>This specifies that the source database is Oracle 10.<br>• SYBASEIQ12<br>This specifies that the source database is Sybase IQ 12.x.<br>• SYBASEASE12<br>This specifies that the source database is Sybase ASE 12.x.<br>• DB2UDB<br>This specifies that the source database is IBM's DB2 UDB.<br>• MYSQL5 |

| | |
|---|---|
| | This specifies that the source database is MySQL 5.x.<br><br>Many other databases have been successfully read and written by SeETL[RT] and the Data Transfer Utility. Most ODBC data sources can be read by the Data Transfer Utility. However, the line of which databases will be supported has to be drawn somewhere and we believe these 7 databases represent the vast majority of the databases that might be used for a data warehouse.<br><br>Default Value:<br>SQLSERVER2005 |
| TargetDatabase | This defines the type of database to which data will be written. Note that MS Access is supported as a target database for the Data Transfer Utility. MS Access is not supported as a target database for SeETL[RT].<br><br>Default Value:<br>SQLSERVER2005 |
| ErrorMessageOutput | This defines where you would like error messages to be written. The valid values and their meanings are as follows:<br>• cerr (Default Value)<br>This specifies that the error messages should be written to the standard C Error output whatever that might be on the server.<br>• ErrorFile<br>This specifies that the error messages should be written to an error file that is specified in the ErrorMessageFileName parameter.<br>• TargetDatabase<br>This specifies that the error messages should be written to the database table accessed via the DBConnectionOutParameter. The table that will be written to is OutCatalogName.OutSchemaName.ctl_message_table.<br><br>Note. If SeETL[RT] is unable to connect to the target database it will be unable to write error messages to the database if you specify TargetDatabase in this option. It is recommended that when performing initial testing error messages are written to cerr or the ErrorFile.<br><br>Default Value:<br>cerr |
| ErrorMessageFileName | This is the fully qualified name of a file that the server has access to that can be opened for writing. This parameter is required if you specify that error messages should be written to the 'ErrorFile'. An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\errormessages.dat<br><br>Default Value:<br>"Not Set" |
| NumberRowsToTransfer | A number of programs have a parameter included called 'NumberRowsToTransfer'. They currently are U001, AT01, AT02, AS01, PR01. This allows the running and testing of programs on a limited number of rows rather than having to process full volumes of data or define views to limit the amount of data that will be presented to the program.<br><br>Valid values are:<br>• Any Numeric Value<br><br>Default Value:<br>0 – meaning do not limit the number of rows to transfer. |
| HeaderRecordExists | The Delimiter Separated Values Reformat Utility reads a flat file which may optionally contain a header row to provide the column names. The HeaderRecordExists parameter indicates the existence of the header record in the input file.<br>Valid values are:<br>• Yes<br>• No |

| | |
|---|---|
| | Default Value:<br>'Yes'. |
| MoveByColumnName | The Delimiter Separated Values Reformat Utility will can move data based on column name or on column position. If the user wants the data to be moved based on Column Name then this flag should be set to 'Yes'.<br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'Yes'. |
| MoveByColumnPosition | The Delimiter Separated Values Reformat Utility can move data based on column name or on column position. If the user wants the data to be moved based on Column Position then this flag should be set to 'Yes'. Note that setting MoveByColumnName = 'No' could have been made the equivalent of setting MoveByColumnPosition = 'Yes'. However, we believe it is more obvious to the user to force the setting of the MoveByColumnPosition parameter to 'Yes' to force move by column position rather than have it be set by a negative answer to another parameter.<br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'No'. |
| DSVDelimiterCharacter | The Delimiter Separated Values Reformat Utility allows the user to specify the delimiter used in the input file by using this parameter. If the input file is not delimited by ',' then this parameter must be used to define the delimiter.<br><br>Note that no validation will be performed on the delimiter passed to the Delimiter Separated Values Reformat Utility. If the value passed is longer then one character it will be ignored.<br><br>Valid Values are:<br>1. Any single ascii character.<br>2. The literal 'tab' for tab delimited files.<br><br>Default Value:<br>','. |
| DelimiterCharacter | SeETL[RT] makes use of the '~' character as the default delimiter. However, this value can be over-ridden by the user to specify any delimiter character. Remember, if you use a delimiter that can appear inside the data in a column the results will be unpredictable and unsupported. Comma delimited is not recommended for data warehousing as street names and house names commonly have commas in them.<br><br>Note that no validation will be performed on the delimiter passed to the SeETL[RT]. It will be assumed that the user knows what he/she is doing if he/she is over-riding the default delimiter.<br><br>Valid Values are:<br>Any single ascii character<br><br>Default Value:<br>'~'. |
| TranslateCharacter1 | The Data Transfer Utility and the Delimiter Separated Values Reformat Utility face the problem that there are only 255 characters in the 8 bit ascii language definition. Therefore it is sometimes hard to make sure that no text (especially user entered text) contains delimiter characters. For example, we have seen phone numbers contain '~~~~' when a user can type into the field!!!!<br>Therefore some mechanism must be provided to allow the translation of one character to another before the input data is inspected for delimiters. If this is |

| | |
|---|---|
| | required the TranslateCharacter1 parameter must be set to invoke the translation process.<br><br>For example, if you use '~' as the delimiter and you need to make sure that there are no '~' characters inside fields coming into the SeETL[RT] processing you can set: TranslateCharacter1=Yes, TranslateCharacter1From=~ and TranslateCharacter1To=X.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'No'. |
| TranslateCharacter1From | This is the input character that will be translated to some other character.<br><br>Valid Values are:<br>Any single ascii character<br><br>Default Value:<br>' ' (blank). |
| TranslateCharacter1To | This is the output character that TranslateCharacter1From is translated into.<br><br>Valid Values are:<br>Any single ascii character<br><br>Default Value:<br>' ' (blank). |
| TranslateNewline | The Data Transfer Utility faces the problem that upstream systems might actually put 'newlines' into the data in the fields. However, both these utilities use 'newline' to delimit the end of the line. In fact the DTU self describing file format uses the newline to delimit records in the file.<br><br>Clearly, newlines cannot be passed around within the SeETL[RT]. In order to allow newslines to be sent to the SeETL[RT] from an ODBC Source the TranslateNewline parameter is provided. When the source data contains a newline you set TranslateNewline=Yes and TranslateNewlineTo=+ (or whatever character you want to represent newline). This will translate all newlines in the input datastream to the '+' character which can then be passed to downstream files.<br><br>When you want to translate the substitute character back to a newline in the data warehouse as you load the data with the DTU you specify TranslateNewline=Yes and TranslateNewlineFrom=+. This will translate the '+' to a newline just before the data is loaded into the data warehouse.<br><br>Note 1. If you want to use non-keyboard characters you can enter the escape character sequence to specify a non display character (\) and pass that character.<br><br>Note 2. You never need to specify both TranslateNewlineFrom and TranslateNewlineTo in one program invocation because the default values of both these fields is the newline.<br><br>Note 3. Those readers who are particularly observant will notice that the DTU feature of being able to 'transfer' files between databases cannot take advantage of the TranslateNewline feature because the substitute character will be loaded as only one translation mechanism is allowed in any one invocation of the DTU. The transfer option actually runs an unload and a load in one run.<br><br>Valid values are:<br>• Yes<br>• No |

| | |
|---|---|
| | Default Value:<br>'No'. |
| TranslateNewlineTo | The character that you want to translate newlines to in order to be able to process newlines within fields inside the SeETL[RT].<br><br>Valid Values are:<br>Any single ascii character<br><br>Default Value:<br>'\n' (newline). |
| TranslateNewlineFrom | The character that you want to translate into newlines in order to be able to load the newline properly into the data warehouse. This parameter is usually provided to the Data Transfer Utility as it is loading data into the Data Warehouse.<br><br>Valid Values are:<br>Any single ascii character<br><br>Default Value:<br>'\n' (newline). |
| DebugLevel | This defines the level of debugging for the run time code. It is expected to be an integer value. The code has a large number of print statements imbedded in it. If you pass a non-zero value as the DebugLevel these print statements will execute and log the program execution at a very detailed level. It is used to assist in debugging and support. You should not set this parameter unless requested to do so by technical support.<br><br>Default Value:<br>0 |
| Audit | This defines whether you would like audit records written for the processing that is performed. The audit records contain the name of the program, the function calling the audit processing, the column name or description of what is being audited and the number of rows being read, inserted, updated, deleted.<br>The audit process is required by SeETL[RT] and so has been included in the Data Transfer Utility. The default value is 'No' and the only valid parameter value is 'Yes'. Audit messages are written to the table OutCatalogName.OutSchemaName.ctl_audit_table.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>"No" |
| InputTableOrFileCanBe Empty | The InputTableOrFileCanBeEmpty parameter allows any program to accept an empty input file and not issue a failure. This is to allow the ETL designer to decide if empty files will be an allowable condition for his/her specific situation.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| AutoCommit | Nearly all ODBC drivers default to AutoCommit='Yes'. This means that every update process is logged at the time the processing is executed. This causes a large amount of commit processing. This is not a problem for relatively small volumes of data. For larger volumes of data it is useful to turn the AutoCommit off. If you are loading significant amounts of data using the Data Transfer Utility (U01) or performing aggregations (AG01) on large amounts of daily data you should set AutoCommit off by setting this parameter to 'No'.<br><br>Valid values are: |

| | |
|---|---|
| | • Yes<br>• No<br><br>Default value:<br>'Yes' |
| CommitFrequency | If you choose to set AutoCommit to 'No' clearly there needs to be a frequency for the commit process. The CommitFrequency parameter allows you to set the number of update/insert/delete operations that will be performed before the next commit. You should consult your DBA for the most effective CommitFrequency for your specific tables and situations. The CommitFrequency may be different for different types of tables. For relatively narrow tables with few columns you might specify a different CommitFrequency to a table with a large number of columns. Note that currently the dimension table processing programs do not allow the CommitFrequency to be changed. It is assumed that they will be performing very little updating on a daily basis because they detect changed rows and only perform updates on rows that have changed.<br><br>Valid values: Any numeric value > 0<br><br>Default value:<br>1,000 |
| UseSequenceName | There are many reasons why one would want to put sequence numbers onto the front of fact tables. The main one being to enable easy fact to fact table links. In order to do this SeETL now supports named sequences for fact table processing. This parameter is used to tell SeETL that a sequence name will be used.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| SequenceName | This parameter defines the name of the sequence that will be used in the processing of a specific fact table. It is recommended to use the name of the view that is used to process the fact table.<br><br>The sequence number requires a field to be available to place the sequence number into. This field is called **`ctl_sequence_num`** and it is a reserved name inside a SeETL ETL Subsystem. This colum must appear on the SeETL view for both the source and the target views. It is recommended that it is simply set to '0' in the SeETL Source View. It will be ignored and overwritten by the attribution process.<br><br>If you request the use of a SequenceName and do not provide the ctl_sequence_num field a mandatory message will be issued and processing will stop.<br><br>An example valid value is:<br>vf_order_lines<br><br>Default Value:<br>"Not Set" |
| RespectSequenceNumbers | The use of sequence numbers and the linking of fact tables to fact tables introduces a specific problem. That is, when a fact table row is updated, the row is given a new sequence number as it is processed. However, because the old sequence number is already placed on many fact tables it is not feasible to force the update of those fact tables.<br><br>The answer to this problem is to retrieve the old sequence number from the old fact table row before the row is replaced.<br><br>This is achieved by using the RespectSequenceNumbers parameter. When set to |

| | |
|---|---|
| | 'Yes' this parameter will force every row being loaded into the target fact table to be checked to see if it already exists. If it does the old sequence number will be written over the new sequence number that was allocated during the attribution processing.<br><br>This parameter is only required when you have identified that some fact records can be updated by fact table processing. If no fact rows can be updated then this parameter should not be used.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| AllowDataConversionErrors | When loading test data for prototyping it is often the case that a small number of rows have some fields in them that contain data that causes a data conversion error. SETL used to terminate on a data conversion error as it was defined to be a severe error.<br><br>Some clients requested that, for initial loading of prototype data that SeETL be updated so that data conversion errors could be detected and allowed to occur via a parameter. We included this parameter to meet the request that data conversion errors be allowed to occur and processing be allowed to continue. An error message, `CTLW0012` , is written to the error message log to warn of the data conversion error.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| ErrorFileName | This is the fully qualified name of a file that the server has access to that can be opened for writing. This is the file that CTLU001 writes error records to when a data conversion error occurs. It is required when AllowDataConversionErrors=Yes and is ignored in all other cases.<br>An example valid value is:<br>d:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlf001.err<br><br>Default Value:<br>"Not Set" |
| ValidateTables | SeETL has been written to perform validation of all table/view names that it is passed in parameters. This is to enable it to give the user the very specific message that the table/view does not exist if a typo was entered for the table/view name.<br>It was always intended that this validation was able to be turned off in Production. We have now implemented this ability to turn off table validation.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| TurboRead | SeETL reads fields from the returned result set from ODBC one field at a time. It is possible for ODBC to return the entire row in one call, or even an array of rows in one call. This parameter tells SeETL to read the major input table of a program one row at a time and not one field at a time. It provides a significant performance boost where the ODBC driver properly supports this calling mechanism.<br><br>Valid values are: |

| | |
|---|---|
| | • Yes<br>• No<br><br>Default value:<br>'No' |
| TranslateDelimterCharacter InQuotes | The Reformat Delimiter Separated Files Utility was upgraded to be able to detect the delimiter character inside quoted strings and translate it to some other character. This parameter tells the Reformat Delimiter Separated Files Utility whether it should invoke this checking.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| TranslateDelimter CharacterTo | The Reformat Delimiter Separated Files Utility was upgraded to be able to detect the delimiter character inside quoted strings and translate it to some other character. This parameter tells the Reformat Delimiter Separated Files Utility what the delimter character should be converted to.<br><br>Valid Values are:<br>Any single ascii character<br><br>Default Value:<br>' ' (blank). |
| TranslateQuoteCharacter InQuotes | The Reformat Delimiter Separated Files Utility was upgraded to be able to detect the quote character inside quoted strings and translate it to some other character. This parameter tells the Reformat Delimiter Separated Files Utility whether it should invoke this checking.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| TranslateQuoteCharacterTo | The Reformat Delimiter Separated Files Utility was upgraded to be able to detect the quote character inside quoted strings and translate it to some other character. This parameter tells the Reformat Delimiter Separated Files Utility what the quote character should be converted to.<br><br>Valid Values are:<br>Any single ascii character<br><br>Default Value:<br>' ' (blank). |
| UseAlternateViewFor SequenceNumberLookup | This parameter takes a little explaining. With the ability to respect sequence numbers on fact records there is an issue. How to look up the previous fact record to retrieve the sequence number to be respected? The initial implementation used the set of integer keys at the front of the fact table as the lookup fields to retrieve the sequence number. However, in the case where any one of these keys change the lookup will fail and the row will not be retrieved correctly. Although it is uncommon for the integer keys to change from one version of a record to another it is far from unheard of.<br><br>To get around this problem we have implemented to ability to provide a view name as an alternate view that can be supplied to perform the lookup to rertrieve the sequence number to be respected.<br><br>Further, to get around the problem of 'how to name the fields' we have also introduced the capability to support 'FuzzyFieldMoves'. The idea of ; |

| | |
|---|---|
| | FuzzyFieldMoves' is that two field names are considered 'equal' if they are identical except for 'pk_' being a prefix of one or the other field.<br><br>By allowing 'FuzzyFieldMoves' and using the alternate view it is possible to define an alternate key using the real key for the record to be able to retrieve the correct sequence number to place on the current record in all situations.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| AlternateViewForSequence NumberLookup | As per the above entry. This parameter is the name of the actual view that will be looked up to retrieve the sequence number to be respected.<br><br>Valid values are: A view name for the lookup<br><br>Default value:<br>'Not Set' |
| AllowFuzzyFieldMoves | As per the entry for above UseAlternateViewForSequenceNumberLookup the AllowFuzzyFieldMoves also has broader implications when used for the Data Transfer Utility. When moving data from a source to a target inside the Data Transfer Utility the AllowFuzzyFieldMoves parameter allows the DTU to considertwo fields equal if the only difference between them is that one of them starts with "pk_". This allows such things as being able to move data from a file/table that does not have the keys defined on it to a target table that does.<br><br>The introduction of AllowFuzzyFieldMoves allows for less coding of views as delimted files or source tables bring data into the SeETL environment.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| RunSegmentationInParallel | The Segmentation Engine is a part of the Application Suite that is being implemented using the SeETL Base Classes and the BI4ALL Data Models.<br><br>The Segmentation Engine can be run in Parallel or sequentially. When the Segmentation Engine is run in parallel it can be run in up to 10 parallel streams, one for each ending digit for the internal integer key used for the customer key in the models. This guarantees that each of the 10 parallel streams contains the same number of customers and that the distribution of customers is relatively random by the ending digit.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| LastDigit_dk_vm_customer | When the parameter RunSegmentationInParallel is set to Yes each instance of the Segmentation Engine must have the LastDigit_dk_vm_customer set to a valid numeric from 0 to 9. This valid numeric will be used to modulus the last digit of the field dk_vm_customer. This allows for the user to parallelise the processing of the Segmentation Engine into up to 10 streams.<br><br>Valid values are: |

| | |
|---|---|
| | • Numeric digit 0 through 9<br><br>Default value:<br>'0' |
| SegmentationGeography WhereClause | The Segmentation Engine allows the user to define a where clause for the geography to be analysed by the segmentation engine. Any where clause can be specified.<br><br>The reason for the SegmentationGeographyWhereClause parameter is as follows. It is expected that the segments that exist within customer bases change across geographic regions below the country level. Therefore it seems reasonable to be able to set the gepgraphic region for a particular segmentation mode to any geographic region requested by the user.<br><br>Note that the totals etc calculated are for that specific geographic region and do not cover the country as a whole. So, should the user wish to have segmentation models for a geographic region and then another for the entire country a separate segmentation model needs to be created for the entire country as well as one for each individual region.<br><br>Valid values are: Any valid where clause for the vm_geography table.<br><br>Default value:<br>'Not Set' |
| SegmentationDemographic WhereClause | The Segmentation Engine allows the user to define a where clause for the demographic group to be analysed by the segmentation engine. Any where clause can be specified.<br><br>The reason for the SegmentationDemographicWhereClause parameter is as follows. It is expected that the segments that exist within customer bases change across Demographic Groups. Therefore it seems reasonable to be able to set the Demographic Group for a particular segmentation mode to any Demographic Groups requested by the user.<br><br>Note that the totals etc calculated are for that specific Demographic Group and do not cover all the customers as a whole. So, should the user wish to have segmentation models for a Demographic Group and then another for the entire customer population a separate segmentation model needs to be created for the entire customer population as well as one for each Demographic Group.<br><br>Valid values are: Any valid where clause for the vm_all_demographic table.<br><br>Default value:<br>'Not Set' |
| SegmentationTypeShort Description | The Segmentation Engine allows the user to define a where clause for the Segmentation Type to be analysed by the segmentation engine. Any where clause can be specified.<br><br>The reason for the SegmentationTypeShortDescription is to allow the user to define their own Segmentation Types. However, PCS will remain the definitive Segmentation Type for Product Catalogue Segmentation.<br><br>Note that the totals etc calculated are for that specific geographic region and do not cover the country as a whole. So, should the user wish to have segmentation models for a gepgraphic region and then another for the entire country a separate segmentation model needs to be created for the entire country as well as one for each individual region.<br><br>Valid values are: Any valid where clause for the vm_all_demographic table.<br><br>Default value:<br>'Not Set' |

| | |
|---|---|
| SegmentationRunDate | The Segmentation Engine allows the user to define the Date on which it is run so that it can be run on a system date that is different to the date considered for the cut off of the Segmentation Engine. This is particularly necessary where transactions may take a number of days to make their way from various locations into the data warehouse.<br><br>The parameter SegmentationRunDate is typically provided in the 'YYYY-MM-DD' format. It must be equal to the format used in the field vm_segmentation_run_log.segmentation_run_date_str. The format of this field is at the discretion of the EDW Architect as it is a string field.<br><br>Valid values are: Any valid date of the format used in the field vm_segmentation_run_log.segmentation_run_date_str.<br><br>Default value:<br>'Not Set' |
| SegmentationRunNumber | The Segmentation Engine allows the user to run the Segmentation Engine many times on the one day. Each run of the Segmentation Engine is numbered sequentially from 1 increasing by one for each run. There is no maximum to the number of times the Segmentation Engine can be run in a single day.<br><br>Later, when data is accessed from the Segmentation Models the SegmentationRunNumber parameter must be correctly set. For example, multiple sets of statistics can be calculated based on the data stored in the Segmentation Models.<br><br>Valid values are: Any valid number though a segmentation run must exist for the where clause to produce a result.<br><br>Default value:<br>'1' |
| StatisticsTypeCode | The Segmentation Engine allows the user to define a codes to be used for the Statistics Types. Therefore execution of the Segmentation Engine must allow for the parameterization of the Statistics Type.<br><br>Valid values are: Any valid Statistics Type as defined in the field vm_stats_type_code.stats_type_code.<br><br>Default value:<br>'Not Set' |
| SegmentationVersion Number | The Segmentation Engine allows the user to define numerous versions of the segmentation models that are then run to produce different versions of output according to the model used.<br><br>The user can run the Segmentation Engine many times on the one day. Each run of the Segmentation Engine is numbered sequentially from 1 increasing by one for each run. There is no maximum to the number of times the Segmentation Engine can be run in a single day.<br><br>Later, when data is accessed from the Segmentation Models the SegmentationRunNumber parameter must be correctly set. For example, multiple sets of statistics can be calculated based on the data stored in the Segmentation Models.<br><br>Valid values are: Any valid number though a segmentation run must exist for the where clause to produce a result.<br><br>Default value: |

| | |
|---|---|
| | '1' |
| NumberPointsToQualify InSegment | The Segmentation Engine allows the user to define the number of points required for qualification into a Segment. This number is applied across all segments for qualification and so the user must decide on a points strategy that calculates a number of points for segmentation membership that is meaningful across all segments.<br><br>This number is then used by CTLSG03 to define segment membership when generating calculations and statistics on top of the sales transactions that have been used to calculate segments.<br><br>Valid values are: Any numeric value<br><br>Default value:<br>'10' |
| SelectTableConnection ToUse | This is something of an 'undocumented feature'. There have been some problems in deadlocking when performing the select to determine the row to be inserted/updated in the Data Transfer Utility.<br><br>The usual issue is deadlocks between the connection for the updates and the connection for the selects. To help the DBA get around these we have introduced the ability to tell the program which connection might be best used to perform the select statement.<br><br>The source code is available and so the DBA can review the source code to know exactly the affect of this parameter. The DBA can decide how to approach the locking for this processing.<br><br>If the UseSelectTableConnectionToUse parameter was not set by the user then default is to use the same connection as the insert, update, delete connections.<br><br>Valid values are: 1 , 2 , 3<br><br>Default value:<br>Changes on usage. |
| UseSelectTableConnection ToUse | If the DBA wishes to influence the connection used to perform selects when the Data Transfer Utility is performing updates or deletes then in order to make the Data Transfer Utulity aware of this desire the DBA must set this parameter to Yes.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| WriteDeleteRecordsToFile | The Data Transfer Utility was upgraded to allow delete records to be written to a file rather than to perform deletes itself. This is then used on conjunction with the Delete Rows from Table Utility CTLU019.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| CTLD001DirectoryName | |
| ReplaceHashWithX InFieldNames | This parameter is in the 'can you believe it' category. We had been doing a bit of AS400 data extraction work and on the AS400 the '#' character is typically used in the field names to mean 'number of'. Of course, if the '#' character was brought into the field names into the populare relational databases the field name would have to be quoted in calls to SQL Server. |

| | |
|---|---|
| | To change the name we usually built a view over tables or performed some other form of translation. Now we have decided to add a parameter to the Delimted File Reformat Utility CTLU005 that will allow this utility to notice that a '#' is being used in a field name in a header record and allow that field name to match a target field name with 'X' in the same position as the '#' in the input field.<br><br>Effectively this parameter allows the user to replace '#' with an 'X' in the target field name without having to do any extra work other than to set the parameter for the processing of the delimited files.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default value:<br>'No' |
| | |

As can be seen from the detailed documentation on parameters, the only specific parameters are those required to specifically identify files being written by the attribution, aggregation and consolidation processes. The way that many files can be processed is by having multiple invocations of each program using different parameters for the input tables/files and output tables/files.

## *8.2. Parameters Specific to Individual Programs*

All parameters are available to all programs. The parameter management routine is imbedded at the start of all program processing so any program has access to any parameter. However, there are some parameters that are only applicable to a single program. These parameters have been documented in this section.

| Parameter Name | Description and valid values |
|---|---|
| ProcessBatchName | ProcessBatchName defines the BatchName to which the ProcessGroup that is going to be processed belongs. It is a parameter to CTLU009.<br><br>The user never needs to set this parameter unless the user decides to run CTLU009 outside the Batch Processing Scheduling Utility.<br><br>Default Value:<br>"Not Set" |
| ProcessGroupName | ProcessGroupName defines the ProcessGroupName that is going to be processed. It is a parameter to CTLU009.<br><br>The user never needs to set this parameter unless the user decides to run CTLU009 outside the Batch Process Scheduling Utility.<br><br>Default Value:<br>"Not Set" |
| BatchSleepSeconds | The Batch Processing Scheduling Utility sleeps between the checks it performs to see if any of the batches now qualify to be started. That is, the Batch Processing Scheduling Utility checks all the pre-requisites at periods of just slightly above every 'BatchSleepSeconds'. Once all Batches are checked and those that qualify are started the Batch Processing Scheduling Utility goes back to sleep.<br><br>Valid Values: Any numeric value. Note that no validation is performed. If the parameter supplied is not numeric zero seconds will be used as a substitute for BatchSleepSeconds.<br><br>Default Value:<br>60 seconds. |
| ProcessGroupSleepSeconds | Process Groups can have 2 types of pre-requisites. It can wait for a previous process group to complete or it can wait for a file to be created. The Process Group Manager sleeps between the checks it performs to see if the pre-requisites have been met. It sleeps for ProcessGroupSleepSeconds seconds.<br><br>Valid Values: Any numeric value. Note that no validation is performed. If the parameter supplied is not numeric zero seconds will be used as a substitute for ProcessGroupSleepSeconds.<br><br>Default Value:<br>60 seconds. |
| DataStageServerName | This is the name of the DataStage Server where the job to run resides.<br><br>Default Value:<br>"Not Set" |
| DataStageProjectName | This is the name of the DataStage Project where the job to run resides.<br><br>Default Value:<br>"Not Set" |
| DataStageUserName | This is the name of the DataStage User that will be used to log onto the DataStage Server where the job to run resides.<br><br>Default Value:<br>"Not Set" |
| DataStageUserPassword | This is the password of the DataStage User that will be used to log onto the |

| | |
|---|---|
| | DataStage Server where the job to run resides.<br><br>Default Value:<br>“Not Set” |
| DataStageJobName | This is the name of the DataStage job to run.<br><br>Default Value:<br>“Not Set” |
| UseDataStageParameter Table | This is a indicator to tell the DataStage job scheduler whether it should use a table/view to read the parameters for a particular invocation of a DataStage job.<br><br>Valid values: ‘Yes’ or ‘No’.<br><br>Default value:<br>‘No’ |
| DataStageParameterTable | If the parameter UseDataStageParameterTable is set to ‘Yes’ then this parameter is the name of the table/view that will be read to retrieve the parameters for the DataStage job. The DataStage job scheduler will read each parameter name parameter value pair and attempt to set the parameter for each parameter read from the table/view. The great benefit of doing this is that jobs can be given their own parameters by job name or can be given parameters from a group of jobs.<br><br>Default Value:<br>“Not Set” |
| SetDefaultField1-10 | The Data Transfer Utility has been upgraded to support the setting of fields in a target table to whatever default values the user would like. The parameters SetDefaultField1-10 allow the user to specify that a field that will be described in subsequent parameters will be set to a default value for this execution of the program. The user must set this parameter before the SetDefaultFieldName or SetDefaultFieldValue will be recognised by the DTU.<br><br>Valid values: ‘Yes’ or ‘No’.<br><br>Default value:<br>‘No’ |
| SetDefaultFieldName1-10 | The Data Transfer Utility has been upgraded to support the setting of fields in a target table to whatever default values the user would like. The parameters SetDefaultFieldName1-10 allow the user to specify the name of the fields on the output table that will be set to at a default value.<br><br>Valid values: Any character string representing a field name.<br><br>Default value:<br>‘Not Set’ |
| SetDefaultFieldValue1-10 | The Data Transfer Utility has been upgraded to support the setting of fields in a target table to whatever default values the user would like. The parameters SetDefaultFieldValue1-10 allow the user to specify the default value to be placed into the SetDefaultFieldName1-10 for every occurrence of this field in the output table.<br><br>Valid values: Any character string representing a valid value for the target field value. Note that embedded blanks are not supported.<br><br>Default value:<br>‘Not_Set’ |
| ReportSafeMetadata Mismatch | The MetaData Checking Utility reports all the MetaData mismatches that it finds. However, many MetaData mismatches are known to be valid and safe. For example, moving a smallint to an integer field is known to be save and valid. The ReportSafeMetadataMismatch flag allows the user to stop messages being generated for MetaData mismatches that are known to be safe and valid. The suer can simply reduce the number of messages to review if he/she would like to not generate these messages. |

| | |
|---|---|
| | As the utility is further developed more MetaData mismatches will be defined to be safe and valid.<br><br>Valid values: 'Yes' or 'No'.<br><br>Default value:<br>'No' |
| AllowDataConversionErrors | A number of clients wanted the ability to load test data even though there may still be data convesion errors in the test data. To facilitate this the Data Transfer Utility now has a parameter called AllowDataConversionErrors. When turned on the Data Transfer Utility will issue a warning message and write the row number of the row that could not be loaded to the warning message and then continue processing.<br><br>This parameter should only be used when testing the loading of data and should not be turned on in a production instance the SeETL. SeETL relies on the data correction utility and the users correction of data prior to the data reaching SeETL and SeETL is not intended to be used as a generalized tool for maintaining cycles of invalid data. It is recommended that this parameter is used with extreme care.<br><br>Valid values: 'Yes' or 'No'.<br><br>Default value:<br>'No' |
| DefaultInvalidDates | We have added the ability to actually validate a date or timestamp as it passes through the Data Validation Utility. This parameter tells the Data Validation Utility to perform this validation.<br><br>Valid values: 'Yes' or 'No'.<br><br>Default value:<br>'No' |
| DefaultInvalidDateTimes | We have added the ability to actually validate a date or timestamp as it passes through the Data Validation Utility. This parameter tells the Data Validation Utility to perform this validation.<br><br>Valid values: 'Yes' or 'No'.<br><br>Default value:<br>'No' |
| DefaultDate | If the user specifies that invalid dates should be defaulted this parameter tells the Data Validation Utility what that value should be.<br><br>Valid values: ISO Standard formatted valid date. YYYY-MM-DD<br><br>Default value:<br>'NULL'. Which will become a NULL in the database. |
| DefaultDateTime | If the user specifies that invalid datetimes should be defaulted this parameter tells the Data Validation Utility what that value should be.<br><br>Valid values: ISO Standard formatted valid date. YYYY-MM-DD HH:MM:SS<br><br>Default value:<br>'NULL'. Which will become a NULL in the database. |
| UseOrderByClause | We discovered, much to our surprise, that SQL Server does not guarantee the ordering of a view even if you put an order by clause of the view. In the face of this we introduced the ability to put an order by clause parameter in the parameter list of programs like CTLU001 and CTLAT01/02 so that the order in which data will be presented can be guaranteed, even on SQL Server.<br><br>Valid values: 'Yes' or 'No'. |

| | |
|---|---|
| | Default value:<br>'No' |
| OrderByClause | This parameter is the corresponding Order By Clause that is to be used by the database to order the data in the desired sequence.<br><br>Valid values: Any order by clause considered valid to be added to the sql statement being executed.<br><br>Default value:<br>'Not Set' |
| SchedulerCanStartBatch | In SeETL We made significant effort to stop accidental double processing. We have utilities called Batch Maintenance Utilities to maintain a semaphore around the batch so that a deliberate act is required to open up the batch for processing. Until now CTLBM01 must have been started by some process outside the Scheduler to allow the scheduler to start processing.<br>One of our large clients requested that we add this capability to the scheduler. That is, allow the scheduler to start the batch. We have included this in SeETL RT 3.0.00. This parameter tells the scheduler that the user has specifically decided that the scheduler can start batch processing based on the WaitFileName parameter that must also be supplied.<br><br>Valid values: 'Yes' or 'No'.<br><br>Default value:<br>'No' |
| SchedulerCanEndBatch | In SeETL We made significant effort to stop accidental double processing. We have utilities called Batch Maintenance Utilities to maintain a semaphore around the batch so that a deliberate act is required to open up the batch for processing. Until now CTLBM02 must have been started by some process outside the Scheduler to allow the scheduler to stop processing.<br>One of our large clients requested that we add this capability to the scheduler. That is, allow the scheduler to stop the batch. We have included this in SeETL RT 3.0.00. This parameter tells the scheduler that the user has specifically decided that the scheduler can stop batch processing based on the WaitFileName parameter that must also be supplied.<br><br>Valid values: 'Yes' or 'No'.<br><br>Default value:<br>'No' |
| | |

## *8.3. Parameters Specific to Memory Mapped IO Processing*

All parameters are available to all programs. The parameter management routine is imbedded at the start of all program processing so any program has access to any parameter. However, there are some parameters that are only applicable to Memory Mapped IO processing. These parameters have been documented in this section.

| Parameter Name | Description and valid values |
|---|---|
| MemoryMappedIODirectory | In early versions of the Memory Mapped IO processing the directory to be used was passed to the programs via an environment variable on Windows and assumed to be /tmp on unix. However, we have upgraded this capability to allow the user to define the directory in which memory mapped files will be placed by using a parameter. This allows memory mapped file directories to be specified in the same way on all platforms.<br><br>The file system that the directory points to should have enough space to retain the entire volume of the memory mapped files. Early testing has shown that AIX does not issue a fail message to the program if there is not enough space on the file group to create the file.<br><br>The name provided must include the final '/' or '\' depending on the operating system. SeETL[RT] assumes that the final '/' or '\' will be supplied and will simply concatenate the rest of the name to this parameter. While this still works it will not provide the result the user desires.<br><br>Default Value:<br>"Not Set" |
| LoadDimensionTableGroup | The first implementation of memory mapped IO provided for a single copy of CTLU012 to load all the dimension tables specified into memory mapped files. Though this worked just fine, when we tested with extremely large dimension tables (10M rows and up) we found that the overall elapsed time was considerable. Further, CTLU012 single threaded to a single CPU and it was the only process running. This seemed like a bit of a waste. So we introduced the LoadDimensionTableGroup parameter to the 'Dim Table Load Control' tab of the Mapping Worksheet and the ctl_dim_table_load_control table.<br><br>The purpose of this parameter is to allow the user to group dimension tables together to load using a single instance of the CTLU012 program. In this way, the user can decide how many instances of CTLU012 are run and which dimension tables are loaded by which instance.<br><br>This provides the maximum level of flexibility for the user to parallelise the loading of dimension tables into memory mapped files as quickly and efficiently as possible depending on other constraints of the machine.<br><br>Default Value:<br>"Not Set" |

# 9. ISSUING COMMANDS TO RUN SEETL

SeETL[RT] uses just one program to achieve each type of processing. For example, just one program is required for all Type1 dimension table maintenance. All attribution processes are handled by one program. This means that the programs require some parameters in order to tell the program what processing to perform. In particular, all programs connect to ODBC sources and read tables, and these parameters must be passed to the programs.

The details of all parameters that can be defined have been documented in section Parameters.

This section demonstrates commands for invoking a suite of programs to load portions of a data warehouse.

The read might be concerned that there are a large number of commands and that one of these commands must be written for each table that is to be processed. This is not so. It is ***highly recommended*** that the user creates a single command that is parameterised to run each type of processing for each combination of input database and target database. This way the user generally only creates one set of commands. For example the user might create one command called RunType1Dimension and then pass parameters to this command to process data appropriately. This mechanism of creating commands in order to run SeETL[RT] has proven a very efficient way to construct large and complex batches. Further, these commands can be called from inside the scheduler further simplifying the actual processing of SeETL[RT].

**(2.1 Enhancement)** In 2.1 it is possible for the user to simply include these commands in the 'Batch Schedule' tab of the SeETL[DT] spreadsheet. It will then be placed into the database using SeETL[DT]. This has made the writing and maintenance of these commands trivial.

## 9.1. CTLDM01 – Type 1 Dimension Maintenance

The following command takes the table SEETL3000.dbo.in_customer_dim and builds it into SEETL3000.dbo.customer_dim as a type 1 dimension table. Notice that this command makes use of the defaulting behaviour of the DBConnectionOutParameter, OutCatalogName, OutSchemaName parameters as the in_customer_dim table is in the same database as the customer_dim table.

This program, and all SeETL[RT] programs will return 0 if the program completed successfully and 1 if the program did not complete successfully.

To load a different type 1 dimension table all you need to do is:

- Copy this command and change the InTableName and OutTableName parameters.
- Set up the dimension maintenance views as described above.
- Specify the levels available in the dimension table and the keys that will be used to specify the unique character string for the level in the dim_table_key_definitions table.

It is that easy. There is not a line of code to write to load a staging table into a dimension table. There will still be code to write to prepare the data for the dimension table which will be site specific.

```
CTLDM01.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=
SEETL3000
InCatalogName= SEETL3000
InSchemaName=dbo
InTableName=in_customer_dim
OutTableName=customer_dim
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes
```

Note 1. For CTLDM11 the CTLF001FileName Parameter must be specified.

## *9.2. CTLDM02 – Type 2 Dimension Maintenance*

The following command takes the table SEETL3000.dbo.in_customer_dim2 and builds it into SEETL3000.dbo.customer_dim2 as a type 2 dimension table.

To load a different type 2 dimension table all you need to do is:

- Copy this command and change the InTableName and OutTableName parameters.
- Set up the dimension maintenance views as described above.
- Specify the levels available in the dimension table and the keys that will be used to specify the unique character string for the level in the ctl_dim_table_key_definitions table.
- Specify the fields in the table that need to be checked for change to generate a type 2 dimension table in the ctl_dim_table_type2_col_defs table.

It is that easy. There is not a line of code to write to load a staging table into a type 2 dimension table.

```
CTLDM02.exe DBConnectionInParameter=DSN=
SEETL3000;SERVER=PETERLAP;UID=sa;PWD=password;DATABASE= SEETL3000
InCatalogName= SEETL3000
InSchemaName=dbo
InTableName=in_customer_dim2
OutTableName=customer_dim2
ErrorMessageOutput=TargetDatabase
DebugLevel=9 Audit=Yes
```

Note 1. For CTLDM12 the CTLF001FileName Parameter must be specified.

## *9.3. CTLAT01 – Attribute a Detailed Input Table*

The following command takes the table SEETL3000.dbo.input_order_facts and builds it into a file that is a detailed fact table load file that will be modelled on the table SEETL3000.dbo.f_order_facts.

CTLAT01 does not actually update f_order_facts. In order to make it re-runnable CTLAT01 performs the attribution process and places the output in a file. That file can then be loaded into f_order_facts using CTLU001, the Data Transfer Utility. This makes CTLAT01 re-runnable with no changes required in the event of some system based failure such as a full disk.

There is one different parameter for this program. It is CTLF001FileName. CTLAT01 writes the attributed fact record to the file named in this parameter. By changing this parameter in various .cmd files it is possible to run multiple attribution processes at the same time.

To attribute a different detailed input table all you need to do is:

- Copy this command and change the InTableName and OutTableName parameters.
- Set up the dimension attribution views if this fact table is going to use dimensions differently to previous fact tables.
- Change the parameter in CTLF001FileName to place the output into a different file. This is highly recommended so that multiple invocations of CTLAT01 do not accidentally over-write each others data.

Again, it is that easy. Should you want to add columns to a fact table you merely change the view of the input table and the view of the output table to include the new column names and the program will send that piece of data through to the attributed fact file as well. There are no code changes required to add new columns to fact tables.

```
CTLAT01.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=sa;PWD=password;DATABASE=SEET
L3000
InCatalogName= SEETL3000
InSchemaName=dbo
InTableName=input_order_facts
OutTableName=f_order_fact
CTLF001FileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF001.DAT
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes
```

## 9.4. CTLAG01 – Aggregate an Attributed Fact Working File

The following command takes the file named in CTLF001FileName table, aggregates it according to the rules specified in the ctl_aggregation_control table and places the aggregated output into the file named CTLF002FileName.

CTLAG01 does not actually use the 'input_order_facts' table however it is recommended that this parameter is left in the command to make it clear which input table is being used as input to the aggregate. CTLAG01 does use the OutTableName (in this case f_order_facts) to look up control information and to generate the names of the sort work tables. You must specify the OutTableName, even though it would seem unnecessary.

Note that CTLAG01 does not perform any database updating apart from the sort work tables. It is re-runnable from the beginning of processing.

To aggregate a different attributed detailed input table all you need to do is:

- Copy this command and change the InTableName and OutTableName parameters.
- Set up the aggregation rules for the summary fact table in the ctl_aggregation_control table.
- Create two new sort work tables to act as temporary space in which to sort/sum the new aggregate. Note that the number of integer keys in the sort work tables is limited to the number of columns that can be placed in the GROUP BY clause of the RDBMS.
- Change the parameter in CTLF001FileName to read the different attributed detailed fact table input file and change CTLF002FileName to place the output into a different file. This is highly recommended so that multiple invocations of CTLAG01 do not accidentally over-write each others data.

```
CTLAG01.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SEE
TL3000
InCatalogName=SEETL3000
InSchemaName=dbo
InTableName=input_order_facts
OutTableName=f_order_facts
CTLF001FileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF001.DAT
CTLF002FileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF002.DAT
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes
```

## 9.5. CTLCL01 – Consolidate Aggregated File with the Summary Fact Table

The following command takes the file named in CTLF002FileName table, compares it to the data already loaded in the 'f_order_facts_summary' table and generate inserts into CTLF003FileName and updates into CTLF004FileName depending on whether the input row already exists in the 'f_order_facts_summary' table.

CTLCL01 does not actually use the 'input_order_facts' table however it is recommended that this parameter is left in the command to make it clear which input table is being used as input to the aggregate.

CTLCL01 does use the OutTableName (in this case f_order_facts) to generate the name of the summary fact table, in this case f_order_fact_summary. You must specify the OutTableName, even though it would seem unnecessary.

Again, CTLCL01 does not actually perform any database updates and it is re-runnable from the beginning should any form of failure occur. It writes all output to CTLF003FileName and CTLF004FileName.

To consolidate a different aggregated file with a summary fact table all you need to do is:

- Copy this command and change the InTableName and OutTableName parameters.
- Change the parameter in CTLF002FileName, CTLF003FileName, CTLF004FileName to read and write the different files.

```
CTLCL01.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SEE
TL3000
InCatalogName=SEETL3000
InSchemaName=dbo
InTableName=input_order_facts
OutTableName=f_order_fact
CTLF002FileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF002.DAT
CTLF003FileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF003.DAT
CTLF004FileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF004.DAT
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes
```

## 9.6. Loading Data into the Data Warehouse

The programs above perform very little actual database updating. To this point in the process the only programs that would have actually updated a database table are the dimension table maintenance programs. All the fact table processing programs send their output to files so that they are re-runnable from the start of processing in the event of some system failure.

It is only after the vast majority of bulk processing has been performed that the data is loaded into the data warehouse tables. This also provides for the easiest fall back position for serious failures. No restore is generally required against the data warehouse database for any failure that occurs prior to the actual loading of the fact tables.

To load the fact tables it was decided to use the File Transfer Utility. This is a generalised utility to unload and load data.

All that is required is that the utility is invoked with the correct parameters. As outlined below.

**Loading a Detailed Fact Table**

The following command will load a detailed fact table. Notice the following special parameters:

- InsertUpdateOption=InsertThenUpdate
  This option is specified given that most records should be inserts. Indeed, if the records should be unique you may want to specify 'InsertOnly'.

- DataMovementOption=Load
  The records in the workfile are loaded.

- WorkFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF001.DAT
  This is the output file from CTLAT01. If you look at this file you will notice that all the aggregate keys are also on the records however they are not on the f_order_fact table. This is possible because the utility program builds up internal information about the target table (f_order_fact) and performs a move by name for all the fields in the source file to the target table.

- LoadFactTable=Yes
  This is a special parameter to tell CTLU001 that it is loading a fact table, as opposed to any other table. When specified CTLU001 will change any column name starting with 'dk_' or 'DK_' to 'pk_' or 'PK_' as appropriate.

```
CTLU001.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SE
ETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=f_order_fact
InsertUpdateOption=InsertThenUpdate
DataMovementOption=Load
WorkFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF001.DAT
LoadFactTable=Yes
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes
```

**Performing Inserts to a Summary Fact Table**

The following command will load summary records into a summary fact table. Notice the following special parameters:

- InsertUpdateOption=InsertOnly
  This option is specified because CTLCL01 has detected that this record does not existing in the summary fact table being processed. If this record is 'suddenly' present it is an error. In the event of a failure and a re-run of this program you may choose to change this parameter to InsertThenUpdate in order to perform inserts and replacements for rows left by the previous running of the program.

- WorkFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF003.DAT
  This is the output file from CTLCL01 that contains insert records.

```
CTLU001.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SE
ETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=f_order_fact_summary
InsertUpdateOption=InsertOnly
DataMovementOption=Load
WorkFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF003.DAT
LoadFactTable=Yes
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes
```

**Performing Updates to a Summary Fact Table**

The following command will perform updates against the summary fact table. Notice the following special parameters:

- InsertUpdateOption=UpdateThenInsert
  This option is specified because CTLCL01 has detected that this record exists in the summary fact table being processed. To minimise processing time the CTLU001 program performs updates against the summary fact table. The load utility does not currently support an UpdateOnly option so UpdateThenInsert is used.

- WorkFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF004.DAT
  This is the output file from CTLCL01 that contains update records.

```
CTLU001.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SE
ETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=f_order_fact_summary
InsertUpdateOption=UpdateThenInsert
DataMovementOption=Load
WorkFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF004.DAT
LoadFactTable=Yes
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes
```

# 10. INSTALLING SEETL RUNTIME

You will have been supplied with a winzip file containing all the components of SeETL[RT]. For windows users the winzip file will contain a windows setup file and the associated required files. For unix users it is just a gzip file. The following instructions are for windows users. Unix users will immediately see the steps that are not related to the unix environment and should just skip those steps.

## 10.1. Step 1 – Unzip the zip file and Install

You need to unzip the winzip file using the extract feature of winzip to maintain the directory structure of the zip file. For windows users it does not matter where you unzip the files to as the installer will ask you for where you want the real files located. Windows users should unzip the zip file to a temporary directory. Unix users should place the unzipped files into the target directory. For both windows and unix users it is recommended that the real target directory is something like:

D:\IBISoftware\SeETL\DesignTime\3.0.00

For the remainder of this section we will call this the **seetlpath** to describe the directories created underneath this directory.

Windows users should run the windows setup file and place the outputs of the setup process into a directory called D:\IBISoftware\SeETL\DesignTime\3.0.00\exe\.

Once you unzip the zip file you should see the following directory structure:

| Directory Name | Directory Description |
|---|---|
| seetlpath\exe\ | This directory contains the executables as well as example commands to run the executables. |
| seetlpath\documentation\ | This directory contains the documentation of the product in PDF format. |
| seetlpath\ddl\filegroup | This directory contains the SQL Server 2000 DDL required to set up the demonstration database and filegroups. |
| seetlpath\ddl\tables | This directory contains the SQL Server 2000 DDL required to set up the demonstration tables. |
| seetlpath\ddl\views | This directory contains the SQL Server 2000 DDL required to set up the demonstration views. |
| seetlpath\data | This directory contains the SQL Server 2000 backup of a test database you can use to test SeETL[RT]. It contains the all the tables and views used in this User Guide. |

**Please Note:**
SeETL[RT] requires that SeETL[RT] Utilities are installed and visible in the PATH of the operating system. You must install SeETL[RT] Utilities separately as they are supplied as a separate non-keyed, non-secure package.

## 10.2. Step 2 – Restore the SQL Server 2005 database SEETL3000

The demonstration database that has been used in this user guide is called SEETL3000. This database in in the seetlpath\data directory as an SQL Server backup file. The database is intended to be placed into the SQL Server directory called

D:\SQLSERVER\SQLDATA\.

If you choose to place the restored database into a different directory you will need to perform a customised restore as documented in the SQL Server 2000 database administration documentation.

## 10.3. Step 3 – Give Userid dba DBA access to SEETL3000

The userid DBA is used in the demonstration commands. To be able to run the commands by name you must create the dba userid, provide the password of 'password' and give the userid DBA access to the demonstration database.

## 10.4. Step 4 – Create ODBC Data Source to SEETL3000

To run the demonstration commands the ODBC data source used is SEETL3000. The ODBC data source has been called the same name as the database itself. There is no necessity for this convention to be used. We have just found it to be useful. In order to run the demonstration commands you must create this ODBC connection.

That's it. You should now be able to run the demonstration scripts as provided. It is recommended you try changing some of the tables and the data in tables to see how the programs react to the changes that you make and for you to become familiar with how to use these new tools.

**Note for Oracle 8/9 users.**

The Microsoft ODBC driver for Oracle 8 is not supported for SeETL[RT]. The documentation on the Oracle ODBC driver is as follows:

*The driver is ODBC 2.5 compliant and supports 32-bit systems. Oracle 7.3x is supported fully; Oracle8 has limited support. The ODBC Driver for Oracle does not support any of the new Oracle8 data types—Unicode data types, BLOBs, CLOBs, and so on—nor does it support Oracle's new Relational Object Model.*

SeETL[RT] is tested and supported with the Oracle9 supplied ODBC driver. It has been used with Oracle 8 extensively and Oracle 8 is known to work. However, Oracle is withdrawing support for Oracle 8 in the not too distant future and Oracle 8 will soon no longer be supported by SeETL[RT].

# 11. LIMITATIONS OR CONSTRAINTS

In order to make SeETL[RT] as easy as possible to implement the software contains a number of limitations and constraints on what you may do. You have already read the constraints on the naming standards of columns that SeETL[RT] needs to know about such as primary keys, date_to, date_from etc.

This section discusses any other limitations not previously documented in a more appropriate place.

## 11.1. General Limitations or Constraints

| Limit Value | Limit Value | Description |
|---|---|---|
| MAX_NUM_DIMENSIONS | 50 | The maximum number of dimension tables that can be attached to a single fact table. |
| MAX_SQL_ERRORS | 10 | The maximum number of SQL errors that can be passed from ODBC back to the error message management routines. If ODBC generates more than 10 messages the later messages will be dropped. |
| MAX_LENGTH_DB_NAMES | 255 | The maximum length for any single component of the three part table name catalog.schema.tablename. |
| MAX_LENGTH_SQL_STMT | 9999 | The maximum length of a generated SQL Statement including insert statements which insert data as parameters. |
| MAX_LENGTH_FQ_FILE_NAME | 1000 | The maximum length of the fully qualified file name passed to any routine. |
| MAX_FIELD_SIZE | 32000 | The maximum size of a single field. ODBC requires any field larger than 32K to be sent to the database using 'chunking'. Such calls are usually for CLOBs/BLOBs. SeETL[RT] does not support the movement of these larger fields. If you need to store CLOBs/BLOBs it is recommended that you pass the key using SeETL[RT] and store these objects in another table. It is also recommended you use another tool to move CLOBs/BLOBs. |
| MAX_FILE_BUFFER | 99999 | The maximum size of the record for input/output from/to any system file. This means that the ascii representation of work files should be less that 100,000 bytes. Since blobs are not yet supported this would seem to be a reasonable limit. |
| MAX_NUM_AGG_KEYS | 9 | The maximum number of aggregate keys allowed on any single dimension table. We have been using 9 for years and never had a problem. If more levels are required the first recommendation is to use a second dimension to immediately give yourself 18 levels. Alternatively we can investigate increasing this limit. |
| MAX_NUM_AGG_JOINED_COLUMNS | 50 | This is the maximum number of columns that can be joined together to form the dimension character key field for a level. We have never used more than 20 so 50 seems to be more than enough. Particularly because the total length of the string is 255 characters. |
| MAX_LENGTH_DIM_CHAR_KEY_FLD | 255 | This is the maximum length of the dimension character key field for a level in a dimension table. This release of SeETL[RT] is the first to allow this level to go past 30 as a standard or 150 as an exception. This has been facilitated by the dimension table lookup routines dynamically allocating the memory for each character string to only be the length of string needed. This has been a major enhancement. |
| MAX_APP_ERRORS | 10 | This is the maximum number of errors an application program can report in one call of the error message class to |

| | | |
|---|---|---|
| | | issue an error message. If more than 10 error messages are required the error message class must be updated and a second set of messages must be issued. |
| MAX_MESSAGE_NUMBER_LENGTH | 20 | The maximum length of an error message number. However the error message numbers have been standardized as 8 characters. |
| MAX_MESSAGE_SEVERITY_LENGTH | 2 | The maximum length of the message severity field. |
| MAX_MESSAGE_LENGTH | 4000 | The maximum length of an error message that may be written to the error message table. |
| MAX_LENGTH_PRIMARY_KEY | 999 | The Generate Delta File Utility compares the Primary Keys of two input files. It was decided to limit the size of the Primary Keys that can be compared. This field defines the limit of length of the primary key. If the full concatenated primary key represented as a character string exceeds MAX_LENGTH_PRIMARY_KEY then an error message will be issued and the program will stop processing. |
| MAX_LENGTH_COMPARE_FIELD | 999 | SeETL[RT] Utilities contain a function that performs case insensitive comparisons of character strings. In order to be case insensitive it always converts characters to upper case. Since it is called in a very large number of cases it was decided to put a limit on the number of characters that can be in a string to be compared. In previous versions of the SeETL[RT] this number was hard coded into the function. It has been decided to set the value as a define for those customers who buy the source code license. |
| MAX_NUM_DEFAULT_FIELDS | 10 | The maximum number of default fields that can be set to some default value in the Data Transfer Utility. |
| MAX_NUM_COLUMNS | 4096 | This is the maximum number of columns allowable on one table or in one file. |
| MAX_NUM_MEMORY_MAPPED_FILES | 255 | The maximum number of memory mapped files that can be opening by CTLU012 at any given time. Please remember that in version 2.1 the number of files required per dimension table on unix is 4. So the total number of dimensional tables that can be loaded into memory on unix in version 2.1 is 63.<br><br>On windows only one file is needed per memory mapped file as semaphores are not implemented in windows. Therefore up to 255 dimension tables can be loaded into memory on a windows machine.<br><br>Please be aware that on windows all memory mapped files must be backed by the system paging file and so there is a total limit to the amount of memory available for memory mapped io on windows of something just less than 4GB. |
| MAX_XML_DOC_ATTRS | 100 | This defines the maximum number of attributes that can be provided to the XML Document Reformatting Utility CTLU018. |
| MAX_TRIES_GET_SEQ_NUM | 10 | This defines the maximum number of tries CTLAT01/02 will perform in order to get a sequence number. If the program tries more times than this it will issue an error message and stop. |
| MAX_NUMBER_PCS_SEGMENTS | 65 | This defines the number of segments for Product Catalog Segmentation. It represents 64 segments plus a totals segment. 64 Segments is used in order to provide a 'chess board' payout of the segments. |

## *11.2. Operating Sytems and Database Supported*

SeETL[RT] has been constantly extended as far as functionality, operating system and database support is concerned over the last few years.

This has led to the following complement of operating systems and databases besing supported at the 2.0 level.

| Operating System | Database |
|---|---|
| Windows 2003 Server<br>(and higher) | Data Warehouse databases supported:<br>• SQL Server 2000+<br>• Oracle 8, 9+<br>• Sybase ASE 12+<br>• Sybase IQ 12+<br>• IBM DB2 UDB 8+<br>• MySQL 5.x+<br>Other ODBC Sources Support:<br>• MS Access<br>Other Sources Supported (Note 1)<br>• Delimited ascii files (any delimiter)<br>• Fixed Format Files<br>• Unformatted ascii files<br>• Unformated binary files (requires technical support to configure the readers) |
| Solaris 9.0 (64 bit) | Data Warehouse databases supported:<br>• Oracle 9+<br>• Sybase IQ 12+ |
| AIX 5.1 (64 bit) | Data Warehouse databases supported:<br>• Oracle 9+<br>• Sybase IQ 12+ |
| Red Hat Linux for Intel | Data Warehouse databases supported:<br>• Oracle 9+<br>• MySQL 5.x+ |

Notes:

1. The tools to read non-ODBC data are supported on all platforms.

2. There is no plan to support 32 bit implementations of Solaris or AIX. If a client wishes to use SeETL[RT] on a 32 bit version of Solaris or AIX support will be performed on a best effort basis.

3. There are no plans to support further databases on Windows platforms. However, should you wish some other database to be tested and supported as a data warehouse database we would be pleased to consider your request and understand your thinking behind your request.

4. Although only MS Access is supported as an additional data source via ODBC in Windows 2000+ we have used and tested many other ODBC sources such as flat files and excel spreadsheets. You may assume that you can use any ODBC data source that complies with ODBC 3.51 or above. However, we do not plan to expend out efforts in supporting every known ODBC source known to mankind. There are simply too many.

5. We have tested and can provide limited support for Solaris/AIX and Oracle/Sybase IQ. We do not have Sun/IBM machines in out development center to develop and support these platforms extensively. For unix platforms we license the source code to the client and the client provides their own 24x7 support with futher support provided by Instant Business Intelligence as needed.

6. The ‘conspicuous exception’ is HP-UX/Oracle. We have simply not had a client running HP-UX/Oracle. We would be pleased to test on this platform and certify it as working correctly should you wish to purchase a license for HP-UX/Oracle and have a machine we can test the SeETL[RT] on. We ported SeETL[RT] to Solaris and AIX in less than one days effort each. SeETL[RT] is written in very standard C++/ODBC code and was intended to be as portable as possible.

## *11.3. MetaData Reports*

Removed as sampled reports are available from the Instant Business Intelligence web site..

http://www.instantbi.com/LinkClick.aspx?link=SeETL+Report+Descriptions.pdf&tabid=59&mid=448
http://www.instantbi.com/LinkClick.aspx?link=SeETL+Example+Reports.zip&tabid=59&mid=448

## *11.4. Other Limitations*

File names passed to SeETL[RT] as parameters may not contain embedded blanks. The embedded blanks will be interpreted as the end of the file name. Passing file names enclosed in quotation marks has been tested and it does work however it is not supported.

**Newlines across Platforms Not Supported**

There always has to be some limitations discovered as a tool is used in the real world. One that was found in SeETL[RT] is this.

When the database is on unix and the data contains newlines and the DTU is running on windows the translation of newlines to some other character does not work. This is because the test inside the code is for the character '\n'.

This means that the newline translation will only work when the source code is compiled on the same platform as the database. It also means that if you are transferring data between unix and windows you must take care to ensure that ftp performs the translation of newlines properly.

This newline translation is so pervasive in ftp tools, and it is so rare for newlines to be embedded inside data, that we are not currently planning to provide the ability for the DTU on windows to perform the newline translation from a unix data source on the fly. If you feel this is a major issue for you please contact us on support@instantBI.com.

# 12. DATA WAREHOUSING UTILITIES

SeETL[RT] uses a suite of Data Warehousing Utilities in performing it's various functions. The main reason the 'utilities' are seen a 'separate' from SeETL[RT] is that they are published on our web site free of charge to anyone who wants to download them. The Data Warehousing Utilities can be thought of as the 'free advertising' for SeETL[RT].

## 12.1. Why Have Data Warehousing Utilities?

Every data warehousing project requires some functions to be available or performed that are very common across many projects. For example:

- The need to transfer data from one location to another:
- The need to know a 'batch processing date' that is separately maintained from the system date.
- The need to protect batches from being accidentally double processed. A not uncommon incident that almost always requires a restore.
- The need to run and re-run many DDL scripts during development.
- The need to be able to easily maintain sets of test data without needing to keep lots of database images, which each need to be maintained in the face of change.

In most cases in the past we have typically written tools/scripts etc for each site or just live with whatever the database manager provided and the local tools of the organisation. Usually these things are pretty inadequate and using them consumes a lot of time that could be better spent actually developing code.

Take for example the simplest task of adding a column to a table and changing another column (that is used in many places) from say varchar (20) to varchar (255). Consider what is involved in this trivial change. You have to change the table DDL, view DDL, test data, ETL code, and try it all out again. Each hand made change introduces the chance for making a mistake.

So, it would be really handy to be able to:

- Make the change to the DDL/View in a place that is really quick.
- Generate all the DDL and run it quickly and easily (as a command).
- Reload the test data for the affected tables.
- Add new values to the new column in the test data.
- Unload the test data and store it away again.
- Make as little change to the ETL as possible.
- Minimise the amount of effort required to re-test the ETL with the new column and changed column and know that it is working.

In response to all these issued we developed these utilities as a part of the development of SeETL[RT]. Since they are very useful outside SeETL[RT] environment we chose to package the utilities separately and make them freely available to any one who wants to use them. This is both a small contribution back to the data warehousing community and a little 'low cost advertising' for SeETL[RT].

Since the utilities are written in C++/ODBC and tested on windows 2000 there will be quite a population of people who could benefit from these tools. If you know other data warehousing professionals who would like to use these tools please feel free to pass along our website address. We recommend that new users download the latest version of the Utilities.

If you know other utilities that would be useful feel free to email us with requests at info@instantBI.com. The ones that have high demand and are really useful we will give consideration to developing. Given the foundation of the sets of classes developed to support SeETL[RT] new utilities are easy and fast to develop and we expect that new utilities will be added on a regular basis.

Best Regards

The Instant Business Intelligence Team

## *12.2. What Utilities are Available?*

The current utilities that are available are as follows:

1. The Data Transfer Utility.
2. The Batch Maintenance Utility
3. The SQL Statement Processor Utility.
4. The SQL Server 2000 DDL Generator Utility.
5. The Oracle 9 DDL Generator Utility.
6. The Delimiter Separate Values Reformat Utility.
7. The Fixed File Format Reformat Utility.
8. The Generate Delta File Utility.
9. The Batch Processing Scheduling Utility.
10. The Process Group Manager Utility.
11. The DataStage Job Submission Utility.
12. The DataStage Job Submission Utility – With Parameters.
13. Load Dimension Tables into Memory Maps Utility.
14. MetaData Checking Utility.
15. MetaData Printing Utility.
16. MetaData Loading Utility.
17. Data Correction Utility.
18. Batch Processing Sleep Utility
19. XML File Reformat Utility
20. Delete Target Rows Utility
21. Bulk Load Rows into SQL Server utility
22. Execute ProcessName From Scheduler Utility
23. The Batch Processing Scheduling Utility. – Resilient Version
24. The Execute SQL from inside database utility.

Each of these utilities will be discussed in the following sections.

## 12.3. What Components are Common across the SeETL[RT]?

SeETL[RT] has a number of components that are common. They are:

1. Parameters and parameter management.
   SeETL[RT] uses a common command/parameter interface for all programs. This means that all programs are able to understand all parameters and no valid parameter passed to a program is rejected as 'invalid' because it is not needed. This means you don't have to worry much about any extra parameters and which ones to delete. Because of this we will document all parameters in one place and only note those parameters that are non-standard with the specific utility.

2. Error Messages and error message management.
   It is not possible to turn off the error messages, as much as some people might want to. You can point the error messages to different outputs as you wish. Keeping track of error messages in important in that they will indicate problems within SeETL[RT]. Because the error messages are common across all components the error messages for the utilities include the error messages for SeETL[RT].

3. Auditing and Audit trail management.
   It is possible to turn auditing off. When auditing is turned on for a program the program will issue a message specifying the number of rows/records read/written for every file/table. This is particularly handy to make sure that all records made it from source to target. It also helps keep track of growth in the number of records flowing into the data warehouse.

4. ODBC access and file access.
   All access to databases or files is via C++ classes. All database accesses are managed such that the calling program is unaware that it is talking to a database apart from the fact that it builds SQL to pass into the class. The same is true for most file access. Further, the fact that the file access uses the same data structures to read/write data files as the ODBC classes use for reading/writing ODBC data sources also means that the classes for database access and file access communicate extremely efficiently through the common data structure. This is a part of the reason for the ease of construction of utilities.

## 12.4. Tables Required in Target Database

If you choose to write error messages and audit messages two tables are required to exist in the target database to receive the messages. These are the ctl_message_table and the ctl_audit_table. Example DDL for each of these tables is provided below.

```
create table dbo.ctl_audit_table
    (  program_name       varchar     (0256)    not null   default 'unknown'
    ,  field_name         varchar     (0256)    not null   default 'unknown'
    ,  total_rows         integer               not null   default 0
    ,  integer_total      integer               not null   default 0
    ,  dollar_total       money                 not null   default 0
    ,  date_time          datetime              not null   default current_timestamp,
    )
;


create table dbo.ctl_message_table
    (  program_name      varchar     (0256)    not null default 'unknown'
    ,  function_name     varchar     (0256)    not null default 'unknown'
    ,  message_number    varchar     (0010)    not null default 'unknown'
    ,  message_severity  varchar     (0001)    not null default 'unknown'
    ,  message_text      varchar     (4000)    not null default 'unknown'
    ,  date_time         datetime              not null default current_timestamp,
    )
;
```

## *12.5. The Data Transfer Utility*

### 12.5.1. Why Have the Data Transfer Utility?

We have been working in data warehousing for some 14 years now. One of the things we have to do time and time again is to move data from one place to another, either as test data or in a re-runnable production environment. There are tools to do this, and the list of them seems to be getting longer. However, they all have their 'features' and 'shortcomings'. The ones with a lot of functionality are expensive and the ones that are cheap have little functionality.

One thing that many of them require is quite a bit of setup work to make it possible to move data from one place to another. For example, a lot of these jobs insist that you specify the movement from source table column to target table column on a column by column basis using the graphical front end tool.

For us, this was fine while we were setting up a re-runnable production job, but when we just wanted to grab hold of some test data we seemed to constantly be fetching data using MS Access and then trying to push it into the target database. We always wanted something easier and this data transfer utility is it.

Lastly, the data transfer utility is used by SeETL[RT] for loading fact tables. As such it has been built with all the 'extras' that are required within SeETL[RT].

### 12.5.2.What Does the Data Transfer Utility do?

As the name implies the Data Transfer Utility moves data from one place to another. Those places are ODBC Data Source Names in either a Windows (win32) or Unix environment or any data source that can be seen from a win32/Unix environment.

The functions performed are as follows:

1. Unload from an ODBC DSN to a self describing work file.
2. Load from the self describing work file into an ODBC DSN using inserts, updates and deletes.
3. Reformat the self describing work file into a database non-specific 'Load Interface File' (LIF).

When loading data into an ODBC DSN you can specify the following behaviour.

1. Insert Only.
   Insert only considers a unique index constraint violation to be an error and will stop processing if a unique index constraint violation occurs.

2. Insert then update.
   In this case a unique index constraint violation causes an update using the table primary key to be used in the where clause.

3. Insert then delete insert.
   This function has been specifically added for Sybase IQ. In Sybase IQ updates are much slower than a delete/insert. And since there is no logging in Sybase IQ the delete/insert does not cause excessive logging. Since it is available for Sybase IQ it has been made available for all databases.

4. Update then insert.
   If the majority of rows are going to exist in the target database performing inserts first is a bit wasteful of processing time for large volumes. Update then insert allows you to issue an update first and then if the record is not found issue the insert.

When reformatting the self describing file into a database non-specific 'Load Interface File' you can specify the following behaviour:

1. Create a Header Row.
   The 'Load Interface File' can be created with a header row with the column names of the target table in the header row.

2. User Quote Characters.
   The 'Load Interface File' can be created with a parameter defined quote character around the column names in the header row and around fields which require a quote character. The fields which will have a quote character wrapped around the data are as follows. If your database manager requires quotes around fields of other data types please contact us on info@instantBI.com. It is a simple exercise to put quote characters around specific data types for specific databases.
   a. SQL_CHAR
   b. SQL_VARCHAR
   c. SQL_LONGVARCHAR

3. Set columns to Null.
   SeETL[RT] is fully Null aware. The way in which most databases manage the loading of nulls is to interpret a specific character sting in the input field as representing that the field should be null. To enable this function you can specify a parameter UseNullCharacter=Yes and NullCharacter=^ to specify that the character '^' will be interpreted as a null in the DTU. You are able to specify any single ascii character as the null character. The DTU substitutes one instance of the null character in the Load Interface File to specify that the field is null. Therefore you must select a character that is not your delimiter, not your quote character, and does not appear in the data as a single character. The default value is the '^' character.

SeETL[RT] does not attempt to generate the database specific load statements. This is because there is such a wide variety of syntax and options across all the databases supported by SeETL[RT].

### 12.5.3. How Does the Data Transfer Utility Determine Primary Keys?

To implement updates and deletes it is obvious the Data Transfer Utility requires knowledge of the primary keys of a table. However, in a data warehouse environment it is generally the case that views are being used as target 'tables'. Also, it is normal that tables have multiple unique index constraints on them that need to be used when performing select/update/delete processing on those tables.

Thus, using the underlying table 'primary key' would prove to be an inadequate method of determining the primary key of a table. This is one reason why ETL tools always allow the definition of keys inside the repository description of a table.

One of the design goals of SeETL[RT] was to minimise the volume of 'metadata' required to control the processing being performed. Thus it is most likely that the Data Transfer Utility will target a 'view' of a table rather than the table itself.

In the Data Transfer Utility primary keys on a target table are indicated by the prefix '**pk_**' (or '**PK_**') in a column name. The Data Transfer Utility assumes that the primary key will be in the first set of columns presented by the view. Making the primary key the Data Transfer Utility must deal with columns other than the first columns is not tested and not supported.

If you specify that the Data Transfer Utility can perform a delete or an update at least one column of the target view/table must be prefixed with 'pk_' or the generated insert/update statement will not contain a valid column in the 'where clause'. This has proven to be a common mistake when first using the Data Transfer Utility.

### 12.5.4. Tables Required in Target Database

- InTableName must be in the source database when performing an 'unload'.
- OutTableName must be in the target database when performing a 'load'.
- InTableName must be in the source database and OutTableName must be in the target database when performing a 'transfer'.
- OutTableName must be in the target database for the generation of the Load Interface File.

### 12.5.5. Why Have a Self Describing File Format for the Data Transfer Utility?

In a number of places within the documentation it has been mentioned that the DTU uses a 'Self Describing File Format'.

In the past the description of the file has been stored separately in a language unrelated to the data itself. It was 'too expensive' to pass the data definitions with the data itself so file definitions were developed and agreed and then programs exchanged data through this 'fixed format' of input file.

The leading ETL tools still suffer from this problem today because they are generally 'column' based and they need the input data defined to them in the format of columns in the repository so that they can read the data. That is, the data definition for the data coming into the leading ETL tools is still stored separately from the data itself. This is also largely driven by the fact that much of the data these ETL tools need to process are not 'self describing files'.

This separation of data definition and data has been the case even as late as the development of DTDs for XML!!!

Some would say that 'finally' the concept of 'self describing files' has started to be widely accepted with languages such as XML and the definition languages XDR and XSD being widely adopted.

When developing SeETL[RT] we investigated the possibility of writing the files being passed between programs as XML files using XSD as the definition language. However, one of the problems with this idea was the 'verboseness' and volume of the data and the amount of CPU processing required to 'decode' the data from XML into a relational format. For implementing a Data Warehouse the cost of this decoding was just out of the question.

So, another, less verbose, less CPU expensive, self describing file format was required. The one we finished up deciding to invent is both simple and elegant. It follows the following rules:

1. All data required to process the file must be inside the file.

2. Nulls must be supported.

3. Newlines delimit the row of data so that humans can also read the file.

4. The format should described the file and data 'efficiently'. The volume of data should not be exploded too much by the self description.

5. The process of encoding and decoding the file should be as efficient as possible.

6. All data is presented as ascii text so that it can be moved between machines/firewalls etc. Since BLOBs are not supported using ascii text was quite ok.

7. The same internal format should be used to support both file information and table information.
   This was very important because it means that the programs are completely unaware as to whether they are reading a file or a table. Also, the ODBC and FILE classes can exchange information between each other quickly and easily. Indeed, transferring data from an ODBC class to a FILE class is as simple as passing a pointer from the ODBC Class to the File class.

Using these rules the self describing file format was defined as described in the following section.

### 12.5.6. What Does the Self Describing File Format Look Like?

The format of the self describing file is described below. The Header Row contains all the definition information required to support the reading and writing of the data to/from a file or ODBC Data Source Name (DSN).

The Header Row defines the following:

1. The number of fields in each row.

2. For each field in the file the Header Row defines:
    a. Field Name, where pk_ is defined to represent a field that participates in the primary key. Note that fields beginning with dk_ and xk_ all have meaning to the DTU and you should avoid naming database fields in the data warehouse with these prefixes.

    b. Primary Key Indicator. Set to 1 if the field is part of the primary key

    c. Field Data Type as defined by ODBC.
    ODBC has a list of data types for fields which are defined by the ODBC standard. The data types are numbered and this number is stored in the file. For example a field which is a '4' is an 'integer' and a column that is a '12' is a varying character string.

    d. Maximum Field Length
    The maximum field length supported in a single ODBC call is 32K. We have set the maximum length of a field in the DTU to be 32,000 bytes. If a field is longer than this length it will be truncated to 32,000 bytes.

    e. Number of decimal digits.
    This is the number of decimal digits that can be stored in the field after the decimal point.

    f. Nullable Indicator
    This field indicates whether the field is defined as being nullable by the database that send the field to the DTU. It is set to '1' if the field is nullable.

The ODBC and FILE classes both understand this data structure. The FILE structure reads this first line to determine the format of the data and to set up internal memory locations to be able to read the data from the file. Further, when a table is being written to a file the ODBC class pointer can be passed directly to the FILE class and it will build this initial line and then data can be passed from the ODBC class to the FILE class and written to the file.

A small example of the file is provided below:

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~ ->
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code ldesc~0~
```

As can be seen from the small amount of sample data there are 4 columns defined and the data in these columns is as follows:

| Column Name | PK Indicator | Data Type | Max Length | Decimal Digits | Nullable? |
|---|---|---|---|---|---|
| pk_row_num | 1 – True | 4 – Integer | 10 | 0 | 0 – False |
| code_value | 0 – False | 12 – Varchar | 10 | 0 | 1 - True |
| code_sdesc | 0 – False | 12 – Varchar | 15 | 0 | 1 - True |
| code_ldesc | 0 – False | 12 – Varchar | 4000 | 0 | 1 - True |

| pk_row_num | code_value | code_sdesc | code_ldesc |
|---|---|---|---|
| 1 | Cvdata | code sdesc | code ldesc |
| 1 | Cvdata | code sdesc | code ldesc |
| 3 | Cvdata | code sdesc | code ldesc |

## 12.5.7.Examples of invoking the Data Transfer Utility

As mentioned the program name is CTLU001 and all parameters are passed via the standard console parameters so common for DOS/Unix commands. The general syntax just being the program name and the parameters stored in a .bat or .cmd file. Note that you cannot use a new line between parameters. The commands below are formatted to make them easier to read.

An example command to transfer the table SEETL3000.dbo.customer_dim to SEETL3000.dbo.customer_dim_test_load1 is as follows:

```
CTLU001.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SEE
TL3000
InCatalogName=SEETL3000
InSchemaName=dbo
InTableName=customer_dim
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SE
ETL3000
OutCatalogName= SEETL3000
OutSchemaName=dbo
OutTableName=customer_dim_test_load1
InsertUpdateOption=InsertThenDeleteInsert
DataMovementOption=Transfer
WorkFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\testdata.DAT
ErrorMessageOutput=TargetDatabase
SourceDataBase=SQLSERVER2000
TargetDataBase=SQLSERVER2000
ErrorMessageFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\testerrormessages.DAT
LoadFactTable=Yes
DebugLevel=0
Audit=Yes
```

However, the following command will also work.

```
CTLU001.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SEE
TL3000
InCatalogName=SEETL3000
InSchemaName=dbo
InTableName=customer_dim
OutTableName=customer_dim_test_load1
InsertUpdateOption=InsertThenDeleteInsert
DataMovementOption=Transfer
WorkFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\testdata.DAT
ErrorMessageOutput=TargetDatabase
```

Notice how many fewer parameters can be provided when use is made of the defaulting behaviour of the parameters. Unfortunately, there is no getting around passing the ODBC DSN connection parameter. It could have been placed in a file however, it seems better to have it as a parameter so that the program can be called from command files accessing a variety of different sources. Since the password must be present in the file obviously read access to these command files should be restricted. Note that the password does not need to be passed in unix implementations of ODBC.

The target table in this case is customer_dim_test_load1. The first part of the DDL to create the table is as follows. Notice that the primary key is named pk_customer_dim. The Data Transfer Utility would also work if this were a view over the target table with a column named pk_customer_dim.

```
create table dbo.customer_dim_test_load1
    (  pk_customer_dim                integer                  not null  primary key
    ,  cust_code                      varchar        (0006)    not null
```

To prepare a database non-specific Load Interface file you may use a command similar to the following:

CTLU001.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SEETL3000
OutCatalogName= SEETL3000
OutSchemaName=dbo
OutTableName=test_table_1
WorkFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ test_table_1.DAT
LoadInterfaceFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ test_table_1.LIF
ErrorMessageOutput=TargetDatabase
LoadFactTable=Yes
CreateLoadImageFile=Yes
DeleteRowToBeLoaded=Yes
UseQuoteCharacter=Yes
UseNullCharacter=Yes
NullCharacter=^
LIFDelimiterCharacter=,
QuoteCharacter=\"

Notes.

1. You must still specify a target table which will be the table that will be loaded. This is required because the load file will be produced as the load image for the target table which does not necessarily have exactly the same columns as the input file.

2. Further, if you specify that the rows to be loaded should be deleted from the target table prior to the load the name of the table is required.

3. If you will be loading the data into a fact table you must still specify LoadFactTable=Yes so that field name differences between the work file and the target table which are required to support fact table processing are accommodated

It is recommended that you carefully review the parameters that are available to be passed to the Data Transfer Utility to fully understand this extended function of the Data Transfer Utility.

The table used in this example is as follows:

```
create table dbo.test_table_1
    (  pk_row_num                              integer  not null
    ,  code_value                              varchar (10)
    ,  code_sdesc                              varchar (15)
    ,  code_ldesc                              varchar (4000)
    )
```

The contents of D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ test_table_1.DAT were as follows:

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~->
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~1~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~1~code ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~1~
```

The Load Interface File D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ test_table_1.LIF produced was as follows:

```
"pk_row_num","code_value","code_sdesc","code_ldesc"
1,"^","code sdesc","code ldesc"
2,"cvdata","^","code ldesc"
3,"cvdata","code sdesc","^"
```

## *12.6. Batch Maintenance Utility*

### 12.6.1. Why Have the Batch Maintenance Utility?

One of the common errors in loading data into a data warehouse is the accidental loading of the same data twice, usually due to the human error of restarting the batch at the beginning of processing rather than at the point of failure. It's an easy mistake to make.

Another requirement which is often overlooked is that for any time variant information, such as type 2 dimensions, there must be a close date on the record being closed and open date on the new record. Many people use the system date. This causes problems in the event that there are multiple days worth of processing on the one day.

For example, if there is a failure in the data warehouse load on a Friday night it will not usually be corrected until the Monday morning. After the correction, the data for Friday, Saturday and Sunday must be processed. If the system date is used the date from and date to ranges on time variant information will not accurately reflect the date that the record was changed.

What is usually required, in a data warehouse that will be loaded on a 'daily' basis is a batch date parameter that can inform the Data Warehouse of the date that is to be used as the 'batch date'.

### 12.6.2. What Does the Batch Maintenance Utility Do?

The Batch Maintenance Utility performs three functions.

1. It checks to see if there is a current batch being run. If there is a current batch being run it ends with an error message and an error condition of 1.

2. If there is no current batch running it updates the default batch date by incrementing it one day as a default. If there is some other change that needs to be made to the batch date it can be made manually.

3. At the end of the batch the Batch Maintenance Utility signals that the batch is completed successfully. Thus allowing a new batch to be processed.

By placing the program CTLBM01 at the beginning of any specific batch for a specific schema and testing for a return code of 0 prior to starting your batch for that schema you protect your processing into that schema from every being doubled up.

By placing program CTLBM02 at the end of processing for any specific batch for a specific schema you will signal the completion of the batch for that schema.

### 12.6.3. Tables Required in the Target Database

The ctl_batch_control table is required to be in the target schema for the batch maintenance utility to perform successfully. In the very first instance, to start processing you must enter an initial record in this table to signal that a batch has 'completed'. This table being empty is considered an 'error' condition because it will only ever be empty prior to any batch being submitted.

The batch_complete_flag is set to 0 when a batch is started and 1 when a batch is completed. Records are never deleted from this table. Thus you will build up a history of the batch processing over the months and years.

```
create table dbo.ctl_batch_control
    (  pk_batch_number                          integer    not null
    ,  batch_date                               datetime   not null
    ,  batch_complete_flag                      integer    not null
    ,  constraint          pk_ctl_batch_control  primary key
       ( pk_batch_number
       )
    )
;
```

### 12.6.4. Examples of Invoking the Batch Maintenance Utility

The Batch Maintenance Utility that is called at startup time is called CTLBM01. It can be invoked as described below. Note that you are expected to provide only 'out' connection information as the ctl_batch_control table is expected to be in the target data warehouse.

CTLBM01.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SEETL3000
OutCatalogName= SEETL3000
OutSchemaName=dbo
OutTableName=ctl_batch_control
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes

The Batch Maintenance Utility that is called at completion time is called CTLBM02. It can be invoked as described below.

CTLBM02.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SEETL3000
OutCatalogName= SEETL3000
OutSchemaName=dbo
OutTableName=ctl_batch_control
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes

## *12.7. SQL Statement Processor Utility*

### 12.7.1. Why Have the SQL Statement Processor Utility?

Is there anything more annoying than having to cut/paste DDL from a file to a window to run a piece of DDL? Especially if you have to run lots of small pieces of DDL, which is often the case during development. We have always preferred to keep each piece of DDL in a separate file and then be able to run DDL as a simple command line interface. So after editing a piece of DDL we could just go to the command line and run it. Or, if we have changed 10 pieces of DDL we can make up a small command file and run all 10 quickly and simply.

All the databases give you some kind of utility to do this. However, they vary from database to database and we feel the error messages from these utilities when they are not working are very hard to use to fix whatever the problem is. So, once we had built the C++ Classes, one of which is able to execute any SQL statement that is supported by the database, it seemed like a great idea to write a small SQL Statement Processor. This way we will only ever have one command interface for all our DDL ever again. A file we can edit and a command we can run. Hooray!!!

### 12.7.2. What Does the SQL Statement Processor Utility Do?

The SQL Statement Processor reads a file and executes SQL Statements found within the file.

Note there are some limitations in the sql statement reader.

- Statements are separated by the semi-colon character (';').
- You may not have components of two commands on one line. When it sees a semi colon it believes this line is the end of the SQL Statement. If you have the start of the next statement on the same line it will also be passed to the statement processor, which will cause an error.
- No validity checking is performed on any statements. They are simply passed to the ODBC data source specified as a parameter and executed.
- If you place a semi colon near the end of the file and then enter blank lines after the semi colon it will interpret this as a blank statement and it will and execute the blank statement. It is recommended that the final semicolon in the file is on the last line.

### 12.7.3. Tables Required in the Target Database

No tables are required in the Target Database to run the SQL Statement Processor.

### 12.7.4. Examples of Invoking the SQL Statement Processor Utility

The SQL Statement is passed into the program using the SQLFileName parameter. Also, since the statement processor is executing statements against a database it was decided to use the 'out' parameters.

```
CTLU002.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SE
ETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=ctl_batch_control
SQLFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\sqlfiletest1.dat
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes
```

## *12.8. The SQL Server 2000 DDL Generator Utility*

Note that the Oracle DDL Generator Utility is almost identical to the SQL Server 2000 DDL Generator Utility. The only point the Oracle DBA should be aware of is that the table and index filegroups in SQL Server 2000 translate into table/index tablespaces in Oracle. The Oracle DDL Generator Utility is CTLU004.

Currently (1/1/2005) there is no other DDL utility generator for other SeETL[RT] supported databases such as DB2, Sybase IQ or Sybase ASE.

### 12.8.1. Why Have The SQL Server 2000 DDL Generator Utility?

Is there anything more tedious than trying to maintain SQL DDL by hand? Perhaps only trying to maintain a lot of views of tables using one of the 'productivity tools' for database design. And who has never had the problem of getting the wrong number of columns in the create view statement when compared to the select statement supporting the view?

(I recall on my very, very first data warehousing project back in 1991 the very first utility I wrote was a small product that allowed me to define all my columns only once and then to specify which columns where in which tables. It generated the DDL and the associated documentation from this little set of files. Saved me hours and hours of tedious ddl changes. Unfortunately it was written in rexx/2, a language that only ran on OS/2 at the time. And I have never bought the version that runs on windows. I always wanted to be able to have that utility again on windows, and so now I have re-written it in C++. )

### 12.8.2. What Does The SQL Server 2000 DDL Generator Utility Do?

The SQL Server 2000 DDL Generator Utility reads a simple model stored in two tables in the target database and from them generates the DDL defined by the data in the tables. It's not meant as a replacement for 'repository' products. It is meant as a quick and easy way of generating DDL for portions of the data warehouse. Especially making sure that the views line up etc.

### 12.8.3. Tables Required in the Target Database

The following tables must be in the target database:

- ctl_ddl_gen_table
- ctl_ddl_gen_columns

**ctl_ddl_gen_table**

The DDL for the ctl_ddl_gen_table table is as follows:

```
create table dbo.ctl_ddl_gen_table
    (  process_row            varchar     (001)   not null default 'n'
    ,  table_name             varchar     (255)   not null default 'unknown'
    ,  view_name              varchar     (255)   not null default 'unknown'
    ,  table_type_ind         varchar     (001)   not null default 'U'
    ,  table_file_group       varchar     (030)   not null default 'unknown'
    ,  index_file_group       varchar     (030)   not null default 'unknown'
    ,  table_ddl_file         varchar     (255)   not null default 'unknown'
    ,  view_ddl_file          varchar     (255)   not null default 'unknown'
    )
;
```

The ctl_ddl_gen_table defines the tables/views that will be generated for this run of the DDL generator. Each record in ctl_ddl_gen_table correlates to one table and one view being generated. The DDL generator always produces the table DDL and the view DDL and places the output into separate files so that you can inspect the DDL prior to running it against the database.

| Field Name | Description |
|---|---|
| process_row | This has the values 'y' or 'n' depending on whether you want the DDL generator to generate the DDL for this table on this run of the DDL generator. |
| Table_name | This is the name of the table that will be generated by the DDL generator. |
| View_name | This is the name of the view that will be generated by the DDL generator. |
| Table_type_ind | This field defines the type of table that will be generated. We recommend you fill in the column as follows:<br>'d' – Dimension table<br>'f' – Fact table<br>'c' – Control table<br>'s' – Staging table<br>'e' – Extract table<br>'w' – Work table<br><br>Note that this field is used by the DDL generator to place the where level_col = 'detail' clause on the tables defined as 'd'imension tables. |
| Table_file_group | This is the SQL Server file group that you would like the physical table to be placed in. You can place the primary index and the data in different file groups if you wish. If you want to partition your fact tables across different file groups you will need to generate the DDL using your usual methods. |
| index_file_group | This is the SQL Server file group that you would like the primary index to be placed in. You can place the primary index and the data in different file groups if you wish. |
| Table_ddl_file | This is the fully qualified name of the file to which you would like to write the DDL for the table. |
| View_ddl_file | This is the fully qualified name of the file to which you would like to write the DDL for the view. |

Some example values for this table are:

| Proc Row? | Table | View Name | Type | Table File Group | Index File Group | Table DDL | View DDL |
|---|---|---|---|---|---|---|---|
| y | f_order_fact | f_order_fact_view | f | seetlfg01 | seetlfg01 | Filename | Filename |
| y | f_order_fact_summ | f_order_fact_summ_view | f | seetlfg01 | seetlfg01 | Filename | Filename |

**ctl_ddl_gen_columns**

```
create table dbo.ctl_ddl_gen_columns
    (  table_name             varchar     (030)   not null default 'unknown'
    ,  table_column_name      varchar     (030)   not null default 'unknown'
    ,  view_column_name       varchar     (030)   not null default 'unknown'
    ,  db_data_type           varchar     (030)   not null default 'unknown'
    ,  col_number_in_table    integer             not null default 0
    ,  col_number_in_pk       integer             not null default 0
    ,  flag_column_nullable   varchar     (001)   not null default 'N'
    )
;
```

The ctl_ddl_gen_columns table defines the columns within each of the tables. Each record in ctl_ddl_gen_columns correlates to column in a table and view.

| Field Name | Description |
|---|---|
| table_name | This is the same table name as specified in ctl_ddl_gen_table. This record is looked up by the DDL generator after it has identified the table as one of the tables for which to generate DDL. |
| table_column_name | This is the name of the column in the table. It will be used in both the create table statement and the create view statement that is generated. |
| view_column_name | This is the name of the column in the view defined to be created in ctl_ddl_gen_table. This name is only used in the generation of the DDL for the view. |
| db_data_type | This is the data type of the column that will be passed to the database in the DDL. It must be a valid data type in value DDL format for the database manager to process. For example, 'integer', varchar (10) and money are all valid data types for SQL Server. |
| col_number_in_table | This is the number of the column within the table. Strictly speaking it is not necessary in the relational environment. However, it is always useful to have columns ordered as you would like to see them. This column can just be an ascending number to place the columns in the table in the order that you would like them. The DDL generator sorts the columns into this order to produce the create table and create view DDL. |
| col_number_in_pk | This is the position of the column in the primary key. If a column is not a part of the primary key the column should be set to 0. Note that the DDL generator is expecting that every table has a primary key. If you want to use the DDL generator to generate the DDL for tables without primary keys you will have to edit the DDL after it is produced.<br>The main reason for this is that it is a rare case indeed in a data warehouse that a table does not have a primary key. All dimension tables have the level column and dim char ky fld as primary keys and all summary fact tables have their combination of integer keys as primary keys. It is recommended that detail level fact tables have primary keys to provide a level of protection against double processing of a file into a fact table. |
| flag_column_nullable | This column is set to 'y' if the column is nullable and 'n' if the column is to be defined as not null. |

Some example values for this table are:

| Table Name | Table Column Name | View Column Name | DB Data Type | Col Number in Table | Col Number in PK | Null? |
|---|---|---|---|---|---|---|
| f_order_fact | pk_time_key | pk_time_key | integer | 1 | 1 | n |
| f_order_fact | pk_prod_key | pk_prod_key | integer | 2 | 2 | n |
| f_order_fact | pk_cust_key | pk_cust_key | integer | 3 | 3 | n |
| f_order_fact | pk_vendor_key | pk_vendor_key | integer | 4 | 4 | n |
| f_order_fact | invoice_qty | invoice_qty | integer | 5 | 0 | n |
| f_order_fact | invoice_amt | invoice_amt | money | 6 | 0 | n |
| f_order_fact | discount_amt | discount_amt | money | 7 | 0 | n |
| f_order_fact | product_code | product_code | varchar (6) | 8 | 0 | n |
| f_order_fact | customer_code | customer_code | varchar (6) | 9 | 0 | n |
| f_order_fact | vendor_code | vendor_code | varchar (6) | 10 | 0 | n |
| f_order_fact | invoice_num | invoice_num | varchar (6) | 11 | 0 | n |
| f_order_fact | invoice_remark | invoice_remark | varchar (30) | 12 | 0 | n |
| f_order_fact | order_date | order_date | datetime | 13 | 0 | n |
| f_order_fact | payment_date | payment_date | datetime | 14 | 0 | n |
| f_order_fact | delivery_date | delivery_date | datetime | 15 | 0 | n |
| f_order_fact_summ | pk_time_key | pk_time_key | integer | 1 | 1 | n |
| f_order_fact_summ | pk_prod_key | pk_prod_key | integer | 2 | 2 | n |
| f_order_fact_summ | pk_cust_key | pk_cust_key | integer | 3 | 3 | n |
| f_order_fact_summ | pk_vendor_key | pk_vendor_key | integer | 4 | 4 | n |
| f_order_fact_summ | invoice_qty | invoice_qty | integer | 5 | 0 | n |
| f_order_fact_summ | invoice_amt | invoice_amt | money | 6 | 0 | n |
| f_order_fact_summ | discount_amt | discount_amt | money | 7 | 0 | n |

### 12.8.4. Example of Invoking The SQL Server 2000 DDL Generator Utility

Note that the SQL Server 2000 DDL Generator does not require an OutTableName because the names of the tables are set in the code.

```
CTLU003.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SE
ETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
ErrorMessageOutput=TargetDatabase
DebugLevel=9
Audit=Yes
```

## *12.9. The Delimiter Separated Values Reformat Utility*

### 12.9.1. Why Have the Delimiter Separated Values Reformat Utility?

The reason for writing this utility is the simplest of all. Customers asked for it. SeETL[RT] was originally written to take data from the staging area (visible via ODCB) and loading it into the Data Warehouse. The process of getting the data from the source system to the staging area was originally outside the scope of what was developed. However, customers wanting to load CSV files into the staging area asked us if we could add such a utility in the same easy to use command line interface as the DTU.

Thus, the Delimiter Separated Values Reformat Utility was born.

### 12.9.2. What Does the Delimiter Separated Values Reformat Utility do?

As the name implies the Delimiter Separated Values Reformat Utility accepts an input file of Delimiter Separated Values, usually comma separated values (CSV), and reformats the file to the DTU self describing format so that the data can then be loaded into the Staging Area for the Data Warehouse.

The Delimiter Separated Values Reformat Utility has the following specific functions:

1. The input delimiter can be provided in the DSVDelimterCharacter parameter. It can be any single ascii character so it is possible to pass the DSVRU any delimited file.

2. It will remove leading and trailing quote characters as defined in the parameter QuoteCharacter which can be any single character although it is recommended that it actually be one of the two quote characters.

3. Where there are three quote characters in a row the DSVRU will return a single quote character rather than remove all three quote characters. So, if a text string must retain a quote character within the string you must put three quote characters together. This is also true if you would like the string to start/stop with a quote character.

   Note that the DSVRU does not require character fields be in quote characters. It handles quote characters because this is how most tools produce CSV files.

4. The format of the self describing file is derived by the target table the DSV file will be loaded into by the DTU. You provide a fully qualified table name and the DSVRU will open this table, retrieve the column information and build an internal structure to hold data to be written to the self describing file ready to be loaded.

5. The method of moving data from the DSV file to the DTU self describing file can be MoveByColumnName or MoveByColumnPosition. That is, if the DSV file contains a header row of column names you can specify HeaderRecordExists=Yes and MoveByColumnName=Yes. The DSVRU will then loop through the input file and detect common column names. If some of the target columns of the table are not in the input file they will be set to NULL.

   If no header record exists then you must move data by column position. You can do this by specifying MoveByColumnPosition=Yes, MoveByColumnName=No and HeaderRecordExists=No. Note that you will have to specify MoveByColumnName=No because this parameter defaults to ‘Yes’ so you must deliberately turn it off in order to set MoveByColumnPosition=Yes. (This is not a mistake.)

6. Once the data is moved from the input record to the output record the output record is written in the standard DSV format to the output file.

### 12.9.3. Tables Required in Target Database

OutTableName must be in the target database.

### 12.9.4. Examples of invoking the Delimiter Separated Values Reformat Utility

The program name is CTLU005 and all parameters are passed via the standard console parameters so common for DOS/Unix commands. The general syntax just being the program name and the parameters stored in a .bat or .cmd file. Note that you cannot use a new line between parameters. The commands below are formatted to make them easier to read.

An example command to reformat the file 'D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1.csv' to the file
'D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1.csv' based on the table SEETL3000.dbo.test_table1 is described below.

Test_table_1.csv contains the following data:

```
"pk_row_num","code_value","code_sdesc","code_ldesc"
1,"cvdata","code sdesc","code ldesc"
2,"cvdata","code sdesc","code ldesc"
3,"cvdata","code sdesc","code + ldesc"
```

Test_table1 is defined as follows:

```
create table dbo.test_table_1
    (  pk_row_num                              integer  not null
    ,  code_value                              varchar (10)
    ,  code_sdesc                              varchar (15)
    ,  code_ldesc                              varchar (4000)
    )
;
```

The command to perform the reformat is as follows:

CTLU005.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=SQL200001;UID=dba;PWD=password;DATABASE=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=test_table_1
CTLF001FileName= D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1.csv
CTLF002FileName= D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1.dat
HeaderRecordExists=Yes
MoveByColumnName=Yes
DSVDelimiterCharacter=,
ErrorMessageOutput=cerr
Audit=Yes

The command produces the following output:

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~->
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

This output can then be loaded directly using the Data Transfer Utility or by using the Data Transfer Utility to create a Load Interface File to then be loaded by the database native load utility.

## *12.10. The Fixed File Format Reformat Utility*

### 12.10.1.Why Have the Fixed File Format Reformat Utility?

Having written the Delimiter Separated Values Reformat Utility we noticed that there was very little effort to adjust the processing of the Delimiter Separated Values Reformat Utility so that it would read 'fixed format files' rather than delimiter separated values. So, though no customers had asked for the utility, it seemed sensible to add it to the list of utilities. Also, fixed format files are a common format of files passed to a Data Warehouse from a mainframe system.

### 12.10.2.What Does the Fixed File Format Reformat Utility do?

As the name implies the Fixed File Formal Reformat Utility accepts an input file which is of 'Fixed Format'. Fixed Format meaning that the rows are of fixed length and the columns are of fixed length. Further, it is expected that no delimiters are present in the data. In fact, the Fixed File Formal Reformat Utility ignores newlines and simply reads chunks of ascii characters. If your fixed format file has newlines in it you must allow the character at the end of the line to read the newline.

Once it has read a row from the input file it moves the data to an internal structure based on the target table that the data will be loaded into. Note that columns can only be moved by column name between the source fixed format file to the target table.

The way that it works is as follows:

- The user specifies a header row of the self describing format in the parameter called CTLF001FileNameFormat. You can generate this heading row by performing an unload on the target table and then editing the header row.
- The Fixed File Formal Reformat Utility reads the header file to determine
    - The name of the columns in the input file.
    - The length of each column in the input file. (Obviously the user must specify the maximum length of each field in the fixed file format file.)
- The Fixed File Formal Reformat Utility then reads the table that the data will be loaded into in the next run of CTLU001 in order to build an internal structure to hold data for the target table in a file called CTLF002Filename. (Note that the Fixed File Formal Reformat Utility does not actually load the data, it merely reformats it so that the data can be loaded by the DTU.)
- The Fixed File Formal Reformat Utility then reads each row of the input file, breaks it up into it's various fields and passes it across to the CTLF002Filename file handler to be written to a DTU self describing file.

In this way the Fixed File Formal Reformat Utility allows the user to bring fixed format files into the SeETL$^{RT}$ to be loaded into the staging area. Of course, the data can be loaded as inserts/updates or it can be further converted into a Load Interface File by the DTU.

The Fixed File Formal Reformat Utility also:

1. Trims leading and trailing blanks for all fixed length fields.
2. Supports the translation of characters from one character to another character. This way a field that you may want to use as a delimiter which appears in the input file can be removed or replaced with a a moer suitable character.
3. Supports the translation of newlines from newline to another character. This way, fixed format files that include newlines can be loaded into the staging area or data warehouse.
4. Supports the setting of Nulls. If the length of the field that is moved to the output character string is 1 and that character is the null character specified in the NullCharacter parameter the input field will be said to eb Null.

### 12.10.3.Tables Required in Target Database

OutTableName must be in the target database.

## 12.10.4.Examples of invoking the Fixed File Format Reformat Utility

The program name is CTLU006 and all parameters are passed via the standard console parameters so common for DOS/Unix commands. The general syntax just being the program name and the parameters stored in a .bat or .cmd file. Note that you cannot use a new line between parameters. The commands below are formatted to make them easier to read.

An example command to reformat the file 'D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1.fff' to the file
'D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1.dat' based on the table SEETL3000.dbo.test_table1 is described below.

The format file Test_table_1.fmt contains the following data. It was generated by downloading Test_Table_1 and then editing with textpad.

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~ ->
code_ldesc~0~12~40~0~1~newline~0~12~1~0~0~
```

Notice that there are 5 columns defined in the format file. The last column is 'newline'. The newline is said to be of type 'varchar' (12) and it is said to be 1 character long, contain no decimal places and be 'not nullable'.

The other columns in the table are the same as the columns in the target table.

Test_table_1.fff contains the following data:

```
123456789012345678901234567890123456789012345678901234567890123456789012345
1         cvdata1  +codesdesc1    +codeldesc1                                +
2         cvdata2  +codesdesc2    +codeldesc2                                +
3         cvdata3  +codesdesc3    +codeldesc3                                +
4         cvdata4  +codesdesc4    +codeldesc4                                +
5         cvdata5  +codesdesc5    +codeldesc5                                +
```

The first line is provided so that you can see how many characters are used by each of the columns in the file. We have also included '+' characters to show you the size of the character string. Each record is delimited by a newline to make the file 'human readable'. However, the newline must be read as a character and allowed for in the input file. The read mechanism used for the Fixed File Formal Reformat Utility is to calculate the length of the record from the format file and then to 'GetSomeChars' equivalent to the record length. It does not use the C++ getline function that relies on newlines being in the data. This means that the fixed format file may/may not have newlines in the data.

When the Fixed File Formal Reformat Utility is run the following output is produced:

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~  →
code_ldesc~0~12~4000~0~1~
4~1234567890~0~1234567890~0~123456789012345~0~67890123456789012345678901234567890 →
12345~0~
4~1~0~cvdata1  +~0~codesdesc1    +~0~codeldesc1                              +~0~
4~2~0~cvdata2  +~0~codesdesc2    +~0~codeldesc2                              +~0~
4~3~0~cvdata3  +~0~codesdesc3    +~0~codeldesc3                              +~0~
4~4~0~cvdata4  +~0~codesdesc4    +~0~codeldesc4                              +~0~
4~5~0~cvdata5  +~0~codesdesc5    +~0~codeldesc5                              +~0~
```

Notice the placement of the '+' characters to show the end of the fixed input fields. Also, notice that the first row has been 'cut up' according to the size of the input fields specified in the format file.

Lastly, test_table1 is defined as follows:

```
create table dbo.test_table_1
    (  pk_row_num                              integer  not null
    ,  code_value                              varchar (10)
    ,  code_sdesc                              varchar (15)
    ,  code_ldesc                              varchar (4000)
    )
;
```

The command to perform the reformat is as follows:

CTLU006.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=SQL200001;UID=dba;PWD=password;DATABASE=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=test_table_1
CTLF001FileName= D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1.fff
CTLF001FileNameFormat= D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1.fmt
CTLF002FileName= D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1.dat
UseNullCharacter=Yes
NullCharacter=^
ErrorMessageOutput=cerr
Audit=No

The command produces the following output:

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~->
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

This output can then be loaded directly using the Data Transfer Utility or by using the Data Transfer Utility to create a Load Interface File to then be loaded by the database native load utility.

## *12.11. The Generate Delta File Utility*

### 12.11.1.Why Have the Generate Delta File Utility?

One of the great problems of building a data warehouse is to have the ability to determine just the records that have changed since the last time the file being processed was read into the data warehouse. Over the years operational systems have improved to be able to provide deltas, but this is only the newer operational systems. Every data warehouse designer must, and usually early in the project, determine how deltas will be determined for each file coming into the data warehouse.

Usually the analysis to determine which files will be sent into the data warehouse as deltas is dependent on the size of the file. Generally, it is much more important to make sure that only the changes to large files like customer and accounts are sent to the data warehouse.

It is often the case that the source system is unable to send just the deltas to the data warehouse for tables such as customers and accounts. In earlier releases of SeETL[RT] this was supported by the fact that both the dimension processing programs are able to detect the fact that no change has occurred to a row in a fact table and will not issue an update if no change has taken place.

However, this kind of support is expensive in processing cycles. It is far more desirable to never send records that have not been changed to the data warehouse.

The other major problem that data warehouse designers face is the detection of deletes. This is a very commonly forgotten problem. We have lost count of the number of projects where the fact a row was deleted could not be detected by the operational system and this fact was also lost on the 'delta' processing.

For example, the early versions of the 'SeETL[RT]' do not support the detection of deletes in the source data. If a source data record is deleted from the operational system or from the staging area the type 2 dimension record is **NOT** updated to reflect an end effective date. That is, the date_to stays stubbornly high saying that the record is still current. This was not a mistake, it is simply not possible to close the record until some row is available to mark the 'delete'.

The Generate Delta file also supports the detection of deletes from the old file to the new file making it possible to be able to do things like close out type 2 dimension records with end effective dates.

We have always wanted a utility that could quickly and easily detected deltas between two files and present those deltas to us as a set of transactions against the first file. We can remember many projects where we have had to write code

However, we have always felt that such a utility was going to be pretty tough to write. However, with all the classes of SeETL[RT] available to us it has turned out to be a relatively easy thing to do. Hence the Generate Delta File Utility was developed.

### 12.11.2.What Does the Generate Delta File Utility do?

As the name implies the Generate Delta File Utility accepts two input files in the Data Transfer Utility Self Describing File format.

- CTLF001FileName:
  This file is the 'old' image of the file that is to be compared to the new image.

- CTLF002FileName:
  This file is the 'new' image of the file that is to be compared to the old image.

The Generate Delta File Utility then produces a file, CTLF003FileName, which is the set of transactions that must have happened to CTLF001FileName in order to produce CTLF002FileName.

This file CTLF003FileName can then be loaded into any ODBC compliant database supported by the Data Transfer Utility and used as input to SeETL[RT].

For example, let's say the users source system is only capable of unloading full copies of the customer record.

It is now possible to:

1. Take the 'unload file' of customer details and convert the customer file from DSV or Fixed Format to the DTU Self Describing File format. (If you are lucky enough to have ODBC access to the source system that contains the customer file you can always just unload the whole customer file.)

2. Compare the previous version of the downloaded and reformatted customer file with the current version of the file to detect the transactions that must have occurred in the system that manages the customer record.

3. Send the changes to the Dimension Processing Process for SeETL[RT].

This ability to detect changes using only file processing will dramatically reduce the data warehousing processing times of data from source systems that are unable to send deltas to SeETL[RT]. In many cases, a Delta generation utility like this makes it feasible to send data to a data warehouse from a source system that cannot determine deltas for itself.

There are some restrictions on this utility to make it easier to use and easier to program.

1. The two input files must have a primary key.
   The utility does not produce reliable results for files where no primary key exists. Each row must have a unique Primary Key.

   This is not as obvious as it sounds. A lot of systems that generate deltas may generate multiple rows, one for each transaction, during the period for which the file is generated.

2. The input files must be sorted in ascending order based on primary key.
   The utility makes use of an algorithm which depends on ascending primary keys to detect changes. It does not perform lookups between the two files.

3. The utility will send it's data to a DTU Self Describing File CTLF003FileName based on the OutTableName parameter passed to the utility.
   a. The OutTableName must exist in the target database when the Generate Delta File Utility is run.
   b. The OutTableName must have a column on it called "delta_ind" (case insensitive). Delta_ind must be defined as varchar(1) or longer and it must be 'not null'.
   c. The "delta_ind" column will contain "I" for Insert, "U" for Update or "D" for Delete for each row depending on the transaction that must have taken place against the old file to produce the new file.

The way that the Generate Delta File Utility works is as follows:

- The program starts up, performs validations, and sets up all in memory working areas.

- The program then reads the first record from each input file.
- While there are rows to process in each input file:
    - Determine if the two rows have the same key
    - If they have the same key then
        - Determine if the data has changed
        - If the data has changed then
            - Write the new row from CTLF002 to CTLF003 and mark it as an update.
        - Read a new row from CTLF001 and CTLF002. (Step through each file)
    - Else
        - If CTLF001 key is less than CTLF002 key then (this is a delete)
            - Write the old row from CTLF001 to CTLF003 and mark it as a delete.
            - Read the next row from CTLF001.
        - Else
            - If CTLF001 key is greater than CTLF002 key then (this is an insert)
                - Write the new row from CTLF002 to CTLF003 and mark it as an insert.
                - Read the next row from CTLF002.

- Once one of the files is at end of file status the program determines which file and processes the remaining rows in the 'non-empty' file accordingly.

- While ( end of file CTLF001 and not end of file CTLF002)
    - Write the new row from CTLF002 to CTLF003 and mark it as an insert.
    - Read the next row from CTLF002.

- While ( not end of file CTLF001 and end of file CTLF002)
    - Write the new row from CTLF001 to CTLF003 and mark it as a delete.
    - Read the next row from CTLF001.

**Important Note 1 – Null Awareness.**

The Generate Delta File Utility is fully 'null aware' and this may cause problems with Oracle databases. If a field is detected to have changed from not null to null (or null to not null) it will be detected as a 'changed' record.

If you compare a source file which has zero length character strings which are not null and you compare them with a file that has been loaded then unloaded from an oracle database with the DTU you may notice that the zero length character strings have changed from not null to null in the loaded/unloaded file.

Comparing the not null zero length character strings with zero length character strings which are null will produce a 'change' record from the Generate Delta File Utility. This problem is caused because when the DTU sends a zero length character string to Oracle using ODBC Oracle interprets the string to be null even if the null indicator passed to Oracle for the field is set to FALSE. This problem is only known to occur on Oracle 8 & 9.

**Important Note 2 – Concatenated Primary Key.**

Comparing values of concatenated primary keys to determine if a row has been inserted or delete is not as easy as it might first appear when using character string interpretations of fields rather than their native data types. SeETL[RT] does not use the native C++ types to store data retrieved from a database. All data is converted to the character string representation of the data.

This has implications on the sort sequence of data provided to the Generate Delta File Utility. The Generate Delta File Utility requires that data is passed to it in ascending order for concatenated primary keys once the elements of these keys have been converted to their C++ character string representations.

This means that a primary key of integer type that is set to 3 turns out to be greater than a primary key of 12 when these two fields are converted to character strings!!!

This is because the integer 3 is converted to the string “3” and 12 is converted to the character string “12”. And when strcmp is used to compare “3” to “12” it says that “12” is less than “3”, which it is.

As a user of the Generate Delta File Utility you need to be aware of how the concatenated primary key string is built so that you can make sure that you present data to it in primary key sequence. Since you can only sort the data in a database or in a DSV file format there is code included in the Generate Delta File Utility to get around this problem in most cases.

However, if you strike one of these cases the symptoms are as follows. You will see extra inserts and deletes generated in the delta file as the Generate Delta File Utility incorrectly determines keys to be less than/greater than they should be. It may even cause problems as the Generate Delta File Utility always reads the row from the file that it thinks contained the key that was of the lower value. In testing this meant that the Generate Delta File Utility walked to the end of one file before then starting to read the remainder of the other file and write inserts or deletes for all remaining rows in the file.

There is not much to be done about this limitation except that you be aware of it and make sure your primary keys are presented to the Generate Delta File Utility in ascending sequence when the comparison will be done by strcmp. The code that has been added to the Generate Delta File Utility to trap most cases is to detect the data types of the fields that tend to get truncated and then pad them with leading zeros out to the full length of the field.

The data types that are considered prone to truncation when converted to character strings are:

- SQL_DECIMAL
- SQL_NUMERIC
- SQL_TINYINT
- SQL_SMALLINT
- SQL_INTEGER
- SQL_BIGINT
- SQL_REAL
- SQL_FLOAT
- SQL_DOUBLE

So, in the example case the integer 3 is defined as being 10 bytes long as a character string and the 3 will be converted to “0000000003”. The 12 is also defined as being 10 bytes long as a character string and it will be converted to “0000000012”. Now, when strcmp is called to compare “0000000003” to “0000000012” it will correctly return that 3 is less than 12 and the Generate Delta File Utility will walk through the two files correctly.

The Generate Delta File Utility also allows a portion of the primary key to be null even though this is not allowed by many databases. If a field is null the value put into the concatenated primary key string is ‘NULL’. This means you should not allow character fields with the value of NULL to be included as part of the primary key if the field can also be null. Again, we do not believe this is a major limitation.

The standard delimiter is also placed between the fields that make up the concatenated primary key to make sure there are no overlaps due to data truncation.

The total length of the concatenated primary key is set to 999 by the defined variable MAX_LENGTH_PRIMARY_KEY. This is because there is a very significant amount of processing that must be

performed for the primary key which is related to the allowable length. 999 characters seems to be more than enough for the primary key of any table we have encountered.

To enable you to understand in full detail how the concatenated primary key is build we have included the code to build the concatenated primary key. Our apologies in advance for the poor formatting when moving the longer code lines to the printed page.

```
BOOL CFILERecordset::GetPrimaryKeyString(char *PrimaryKeyString)
{

int          index1 = 0 ;
int          index2 = 0 ;
int          ws_lengthPrimaryKeyString ;
char         ws_work_field1[MAX_FIELD_SIZE] ;
char         ws_work_field2[MAX_FIELD_SIZE] ;
int          ws_length_field_data = 0 ;
int          ws_number_zeros_to_pad = 0 ;

      strcpy(PrimaryKeyString, "") ;

      ws_lengthPrimaryKeyString = 0 ;

      for (index1 = 0 ; index1 < number_result_columns ; index1++ )
        BEGIN_
            IF (ptr_table_field_desc_array[index1].primary_key EQUALS_ TRUE ) THEN
              BEGIN_
                IF (ptr_table_field_desc_array[index1].field_is_null EQUALS_ TRUE)→
                                                                          THEN
                    BEGIN_
                      ws_lengthPrimaryKeyString = ws_lengthPrimaryKeyString + 4 ;
                        IF (ws_lengthPrimaryKeyString > MAX_LENGTH_PRIMARY_KEY) THEN
                          BEGIN_
                              strcpy(PrimaryKeyString, "Primary Key Greater than→
                                     MAX_LENGTH_PRIMARY_KEY") ;
                              CALL_RETURNED_OK = FALSE ;
                              return CALL_RETURNED_OK ;
                          END_
                        ELSE
                          BEGIN_
                            strcat(PrimaryKeyString,"NULL") ;
                            strcat(PrimaryKeyString,ws_delimiter) ;
                          END_
                    END_
                  ELSE
                    BEGIN_
                      ws_lengthPrimaryKeyString = ws_lengthPrimaryKeyString + →
                        (int) strlen(ptr_table_field_desc_array[index1].field_data) ;
                        IF (ws_lengthPrimaryKeyString > MAX_LENGTH_PRIMARY_KEY) THEN
                          BEGIN_
                              strcpy(PrimaryKeyString, "Primary Key Greater than →
                                    MAX_LENGTH_PRIMARY_KEY") ;
                              CALL_RETURNED_OK = FALSE ;
                              return CALL_RETURNED_OK ;
                          END_
                        ELSE
                          BEGIN_
                            strcpy(ws_work_field1 , →
                                   ptr_table_field_desc_array[index1].field_data ) ;
                              IF ((ptr_table_field_desc_array[index1].field_type→
                                                          EQUALS_ SQL_DECIMAL) OR_
                                   (ptr_table_field_desc_array[index1].field_type→
                                                          EQUALS_ SQL_NUMERIC) OR_
                                   (ptr_table_field_desc_array[index1].field_type→
                                                          EQUALS_ SQL_SMALLINT) OR_
                                   (ptr_table_field_desc_array[index1].field_type→
                                                          EQUALS_ SQL_INTEGER) OR_
```

```
                                        (ptr_table_field_desc_array[index1].field_type→
                                                        EQUALS_ SQL_REAL) OR_
                                        (ptr_table_field_desc_array[index1].field_type→
                                                        EQUALS_ SQL_FLOAT) OR_
                                        (ptr_table_field_desc_array[index1].field_type→
                                                        EQUALS_ SQL_DOUBLE)) THEN
                                    BEGIN_
                                      ws_length_field_data = →
                    (int) strlen (ptr_table_field_desc_array[index1].field_data) ;
                                      ws_number_zeros_to_pad = →
                                   ptr_table_field_desc_array[index1].field_length - →
                                              ws_length_field_data ;
                                     strcpy(ws_work_field2, "") ;
                                     for (index2 = 0 ; →
                                              index2 < ws_number_zeros_to_pad ; index2++)
                                       BEGIN_
                                         strcat(ws_work_field2,"0" ) ;
                                       END_
                                     strcat(ws_work_field2 , →
                                       ptr_table_field_desc_array[index1].field_data) ;
                                     strcpy(ws_work_field1 , ws_work_field2 ) ;
                                   END_
                                  strcat(PrimaryKeyString,ws_work_field1) ;
                                  strcat(PrimaryKeyString,ws_delimiter) ;
                              END_
                         END_
                    END_
               END_
          END_

      CALL_RETURNED_OK = TRUE ;
      return CALL_RETURNED_OK ;

}
```

### 12.11.3.Tables Required in Target Database

OutTableName must be in the target database.

## 12.11.4.Examples of invoking the Generate Delta File Utility

The program name is CTLU007 and all parameters are passed via the standard console parameters so common for DOS/Unix commands. The general syntax just being the program name and the parameters stored in a .bat or .cmd file. Note that you cannot use a new line between parameters. The commands below are formatted to make them easier to read.

An example command to compare the two files:

- D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1_V01.dat and
- D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1_V02.dat

  to produce

- D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \ test_table_1_delta.DAT

The table test_table1delta is defined as follows:

```
create table dbo.test_table_1_delta
    (  pk_row_num                              integer  not null
    ,  code_value                              varchar (10)
    ,  code_sdesc                              varchar (15)
    ,  code_ldesc                              varchar (4000)
    ,  delta_ind                               varchar (1) not null
    );
```

The command to perform this comparison is as follows:

CTLU007.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=SQL200001;UID=dba;PWD=password;DATABASE=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=test_table_1_delta
CTLF001FileName= D:\CTLOR01\RELEASE\data \test_table_1_V01.dat
CTLF002FileName= D:\CTLOR01\RELEASE\data \test_table_1_V02.dat
CTLF003FileName= D:\IBISoftware\SeETL\DesignTime\3.0.00\Data \test_table_1_delta.dat

Some samples of test data and the results are as follows. They are presented in some details so that you can understand what the utility does in each case.

**Example 1.**

Change the value of some fields in the data.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata1~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~xcode + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~2~0~cvdata1~0~code sdesc~0~code + ldesc~0~U~0~
5~3~0~cvdata~0~code sdesc~0~xcode + ldesc~0~U~0~
```

**Example 2.**

Change some of the field values to null even though the data values do not change.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~1~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~1~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~2~0~cvdata~0~code sdesc~1~code + ldesc~0~U~0~
5~3~0~cvdata~0~code sdesc~1~code + ldesc~0~U~0~
```

**Example 3.**

Delete a row from the data. The row with the primary key of 3 is deleted.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~3~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
```

**Example 4.**

Insert a row into the data. In this case the row with primary key of 3 is inserted.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~3~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
```

**Example 5.**

Delete a number of rows at the start of the file. This is to provide the read startup works. The first three rows are deleted from the input file.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~5~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~5~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~1~0~cvdata~0~code sdesc~0~code ldesc~0~D~0~
5~2~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
5~3~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
```

**Example 6.**

Insert some new rows at the start of the file. Again, this is to demonstrate the startup reading. The first three rows are inserted.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~5~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~5~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~1~0~cvdata~0~code sdesc~0~code ldesc~0~I~0~
5~2~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
5~3~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
```

**Example 7.**

Delete rows from the end of the file. This is to demonstrate the correct processing of end of files. The last 4 rows are deleted.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~7~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~8~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~9~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~10~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~11~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~12~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~7~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~8~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~9~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
5~10~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
5~11~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
5~12~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
```

**Example 8.**

Insert new rows at the end of the data. This is to demonstrate end of file processing. The last 4 rows are inserted at the end of the file.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~7~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~8~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~7~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~8~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~9~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~10~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~11~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~12~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~9~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
5~10~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
5~11~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
5~12~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
```

**Example 9.**

Delete four rows within the file. The rows deleted are 3, 6, 9 and 12.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~5~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~6~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~7~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~8~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~9~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~10~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~11~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~12~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~5~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~7~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~8~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~10~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~11~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~3~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
5~6~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
5~9~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
5~12~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
```

**Example 9.**

Insert four rows into the file. In this case the rows inserted are 3,6,9 and 12.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~5~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~7~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~8~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~10~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~11~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~5~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~6~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~7~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~8~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~9~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~10~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~11~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~12~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~3~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
5~6~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
5~9~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
5~12~0~cvdata~0~code sdesc~0~code + ldesc~0~I~0~
```

**Example 10**

Changing of a primary key of a row in a table.

Note that in the following example that the field code_value has also been made part of the primary key. This is to demonstrate the effect of changing a primary key of a record.

**test_table_1_V01.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~1~12~10~0~1~code_sdesc~0~12~15~0~1~ →
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~5~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_V02.dat**

```
4~pk_row_num~1~4~10~0~0~code_value~1~12~10~0~1~code_sdesc~0~12~15~0~1~ →
code_ldesc~0~12~4000~0~1~
4~1~0~cvdata~0~code sdesc~0~code ldesc~0~
4~2~0~cvdata1~0~code sdesc~0~code + ldesc~0~
4~3~0~cvdata~0~code sdesc~0~xcode + ldesc~0~
4~4~0~cvdata~0~code sdesc~0~code + ldesc~0~
4~5~0~cvdata~0~code sdesc~0~code + ldesc~0~
```

**test_table_1_delta.dat**

```
5~pk_row_num~1~4~10~0~0~code_value~0~12~10~0~1~code_sdesc~0~12~15~0~1~→
code_ldesc~0~12~4000~0~1~delta_ind~0~12~1~0~1~
5~2~0~cvdata~0~code sdesc~0~code + ldesc~0~D~0~
5~2~0~cvdata1~0~code sdesc~0~code + ldesc~0~I~0~
5~3~0~cvdata~0~code sdesc~0~xcode + ldesc~0~U~0~
```

## *12.12. Batch Processing Scheduling Utility*

### 12.12.1. Why Have the Batch Processing Scheduling Utility?

SeETL[RT] was first released to be run as a set of commands executed from the command line in windows/unix. The intention was to integrate SeETL[RT] processing with the customers existing scheduling facilities. However, a number of customers have asked for a scheduling utility that is more aware of the parallel processing requirements of data warehousing ETL processes and that is simple to set up.

A second requirement that has been requested by a number of customers is some form of GUI with which to define and manage the batches of jobs required to maintain the data warehouse.

As a result, the Batch Processing Scheduling Utility has been developed.

The Batch Processing Scheduling Utility will be one of the 'steps' to implementing the GUI to manage batch processes. Once the functionality of the Batch Processing Scheduling Utility has stabilised it will be easier to develop the GUI.

Since SeETL[RT] is a command line based application it was just as easy to generalise the scheduler so that it could execute any command line based application rather than just SeETL[RT]. Thus the Batch Processing Scheduling Utility can schedule and manage the processing of any command line based application on window2000 or unix.

## 12.12.2.Tables Required in Target Database

The Batch Processing Scheduling Utility requires a number of tables be defined in the data warehouse database. Or in any ODBC compliant database if you just want to use it as a scheduler. For example, it would be possible to run the scheduler just using Microsoft Access as a small database to hold the schedule and to manage the batch processing. However, it is recommended that you use a relatively robust database manager that contains row level locking.

### *12.12.2.1.The Batch Control Table*

The Batch Processing Scheduling Utility works in conjunction with the Batch Maintenance Utility. The ctl_batch_control table acts as a semaphore for batch processing. No batch processes can be run against the data warehouse unless the 'batch_complete_flag' is set to 0=False.

The Batch Processing Scheduling Utility will wait until such time as the 'batch_complete_flag' is set to 0=False before checking any other constraints. If no record exists in this table then the Batch Processing Scheduling Utility will not function.

```
create table dbo.ctl_batch_control
    (  pk_batch_number                          integer     not null
    ,  batch_date                               datetime    not null
    ,  batch_complete_flag                      integer     not null
    ,  constraint          pk_ctl_batch_control  primary key
       ( pk_batch_number
       )
    )
;
```

#### *12.12.2.2.The Batch Pre-Requisite Table.*

The ctl_batch_pre_req table is defined as follows in SQL Server 2000. It can be created in any ODBC compliant database with similar data types for each field.

```
create table dbo.ctl_batch_pre_req
    (  pk_process_batch_name        varchar (255)     not null    default 'unknown'
      ,pk_pre_req_number            integer           not null    default 0
      ,pre_req_type                 varchar (020)     not null    default 'unknown'
      ,file_exists_file_name        varchar (4000)    not null    default 'unknown'
      ,day_of_week                  varchar (0020)    not null    default 'unknown'
      ,day_of_month                 integer           not null    default 0
      ,start_hour                   integer           not null    default 0
      ,start_minute                 integer           not null    default 0
      ,start_timestamp              datetime
      ,short_description            varchar (4000)    not null    default 'unknown'
      ,long_description             varchar (4000)    not null    default 'unknown'

    )
;
```

The following table explains the values of each column and the functions available.

| Column Name | Function |
|---|---|
| pk_process_batch_name | This is the batch name assigned for the specific batch. It can be any character string up to 255 characters. It is recommended that the batches are given some form of meaningful name that sorts by the batch names.<br><br>Note that the batch name is not associated with the pk_batch_number in the ctl_batch_control table. The pk_batch_number is used to record sequential numbers of batches, usually based on daily batch processing. The pk_process_batch_name is the name of a batch that will be executed on some form of scheduled basis. This means the same pk_process_batch_name may be run many times and have many pk_batch_numbers recorded against it.<br><br>Note also that Oracle treats data fields as case sensitive so the value assigned to pk_process_batch_name must be in the same case across the different tables in which it is used if you are using Oracle as the database for the scheduler.<br><br>Example: batch001.<br><br>Default: 'unknown' |
| pk_pre_req_number | A batch may have many pre-requisites. All of which must be true/passed prior to the start of the batch. For example, it might be desirable to say that a batch must not start until a specific file is present **and** a certain time of day has been reached.<br><br>The Batch Processing Scheduling Utility does not currently support the 'or' operator for multiple pre-requisites.<br><br>This is the number of one or more batch pre-requisites. It is a sequential number used to sort the pre-requisites of a batch. The pre-requisites will be testing in the order they are presented. If a pre-requisite fails the later pre-requisites for the batch are not tested.<br><br>Examples: 1, 2, 3, 4 etc.<br><br>Default: 0 |
| pre_req_type | There are a number of types of pre-requisites that can be defined for a batch. These pre-requisite types are used in conjunction with the other fields on this table to determine the exact pre-requisite for the starting of a batch. |

The types and their descriptions are as follows:

| Pre-Requisite Type | Description |
|---|---|
| FileExists | This pre-requisite row indicates that a specific file must exist prior to the batch being started. The full path name of the file to check is in field file_exists_file_name.<br><br>The file is checked using the fopen statement so the string in the field file_exists_file_name must be valid to be passed as a parameter to fopen. |
| Daily | This pre-requisite row indicates that the batch will be run daily at a specific time. The time for the batch to start is defined in start_hour and start_minute. When the system clock goes past the start hour every day the batch will be considered to pass the pre-requisite. |
| Weekly | This pre-requisite row indicates that the batch will be run on a specific day of the week at a specific time.<br><br>The day that the batch will start in is stored in the field day_of_week.<br><br>The time for the batch to start is defined in start_hour and start_minute. When the system clock goes past the start hour on the day of week specified the batch will be considered to pass the pre-requisite. |
| Monthly | This pre-requisite row indicates that the batch will be run on a specific day of the month at a specific time.<br><br>The day that the batch will start in is stored in the field day_of_month which is an integer rather than a character string.<br><br>The time for the batch to start is defined in start_hour and start_minute. When the system clock goes past the start hour on the day of the month specified the batch will be considered to pass the pre-requisite.<br><br>Note that if the intention is to start the batch on the last day of the month the value 99 can be used for the day of the month. Leap years are not currently supported. |
| OneTime | It is possible that you would like to run a batch only once and would like to schedule it to run at a certain time rather than have to submit it yourself. For example, during testing this would be quite handy to make sure that the batch does not accidentally get run a second time.<br><br>For a 'OneTime' run the timestamp to start the job is stored in start_timestamp. When the system clock goes past the start_timestamp the batch is said to pass this pre-requisite. |

| | |
|---|---|
| file_exists_file_name | This is the fully qualified pathname of the file that must exist before the batch will be started when the pre-requisite type is set to 'FileExists'. |
| day_of_week | This is the day of the week that the batch will be started when the pre-requisite type is set to 'Weekly'.<br><br>Valid values are:<br>Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday. |

| | |
|---|---|
| day_of_month | This is the day of the month that the batch will be started when the pre-requisite type is set to 'Monthly'.<br><br>Valid values are:<br>1-31, and 99.<br><br>Note that 99 is interpreted as 'last day of the month'. Leap years are currently not supported. |
| start_hour | When the pre-requisite type is set to 'Daily', 'Weekly or 'Monthly' this field stores the hour in which the batch will start. The scheduler uses the 24 hour clock.<br><br>To start a batch in the hour after 6am this field is set to 6. To start a batch in the hour after 6pm this field is set to 18. |
| start_minute | When the pre-requisite type is set to 'Daily', 'Weekly or 'Monthly' this field stores the minute of the hour at which the batch will start.<br><br>To start a batch at 6:06 am you would specify start_hour=6 and start_minute=6. The batch will be started in the first check of the pre-requisites to occur after the system clock goes past 6:06am.<br><br>To start a batch at 6:18pm you would specify start_hour=18 and start_minute=18. The batch will be started in the first check of the pre-requisites to occur after the system clock goes past 6:18pm. |
| start_timestamp | When the pre-requisite type is set to 'OneTime' this field contains the timestamp at which the batch should start. The batch will start in the first check of per-requisites after the system clock passes the timestamp. |
| short_description | This column contains a short description of the Batch that will be printed onto the reports for the batch processing. It is intended to assist the support personell understand the batches a little more easily. |
| long_description | This column contains a long description of the Batch that will be printed onto the reports for the batch processing. It is intended to assist the support personell understand the batches a little more easily. |

Each row in this table represents a batch of programs that can be scheduled to be run at certain points in time or when certain files exist on the machine.

Note that pre-requisites can be combined so that it is possible to force a batch to wait until a certain point in time and a certain file exists on the system. This way it is possible to not start a batch early because a signal file has been placed onto the system. This is often the case when data is being sent to the data warehouse server by a system you have no control over but you do not want to start batch processing until after hours.

#### *12.12.2.3.The Process Group Pre-Requisites Table*

Setting up batches does not actually allow you enough control to make full use of parallel processing because it is not possible to run multiple batches at a time and to ensure that multiple submitted batches run to conclusion before starting another batch.

To enable such control of parallel processing the Batch Processing Scheduling Utility has implemented the concept of:

- Process groups and
- Processes (also called commands).

The following table is the Process Group Pre-Requisite definition. Most importantly, a Process Group can have another Process Group as a pre-requisite. This allows the user to build semaphores out of process groups so that many process groups can be dependent on one process group.

```
create table dbo.ctl_proc_grp_pre_req
    (  pk_process_batch_name        varchar (255)     not null    default 'unknown'
      ,pk_process_group_name        varchar (255)     not null    default 'unknown'
      ,pk_pre_req_number            integer           not null    default 0
      ,pre_req_type                 varchar (020)     not null    default 'unknown'
      ,file_exists_file_name        varchar(4000)     not null    default 'unknown'
      ,pre_req_process_group        varchar (255)     not null    default 'unknown'
      ,short_description            varchar (4000)    not null    default 'unknown'
      ,long_description             varchar (4000)    not null    default 'unknown'
)
;
```

The following table explains the values of each column and the functions available.

| Column Name | Function |
|---|---|
| pk_process_batch_name | This is the batch name assigned for the specific batch. It is the same batch name as occurs on the ctl_batch_pre_req table. A Process Group is placed into a batch by using the same name that defines the batch.<br><br>Example: batch001.<br><br>Default: 'unknown' |
| pk_process_group_name | This is the name of the Process Group. A Process Group, as the name implies, groups together a set of Processes (or commands) that will be run sequentially. That is, within the one process group, each Process must run one after the other and each process must finish with a zero return code.<br><br>Any process that finishes with a return code other than zero will be considered to be abnormally ended (abended). When a Process abends the Process Group and the Batch will be marked as having been abended and the user must 'Restart' the batch in order to complete it. Just re-submitting the batch will not re-start the batch.<br><br>If the user just re-submits the batch the scheduler will issue a message to say the user re-submitted an abended batch and a restart is required. This is to ensure that some user intervention takes place after a failed Process.<br><br>Process group names must be unique within a specific batch.<br><br>Example: group001, group002, group003 etc. |
| pk_pre_req_number | This is the number of the pre-requisite for the process group. A process group can have as many pre-requisites as the user likes. The pre-requisites will be tested in the order of the pre-requisite numbers. |

| | |
|---|---|
| | The later pre-requisites will only be tested when the earlier pre-requisites are passed. |
| pre_req_type | This field defines the type of pre-requisite for the process group. A Process Group may only have 2 types of pre-requisites. ProcessGroup and FileExists: FileExists is a little redundant because it is available at the batch level, however, it seems like a useful pre-requisite type to have for a process group.<br><br>By making a number of Process Groups dependent on another Process Group and then making another subsequent Process Group dependent on the group of Process Groups you can construct parallel processing batches.<br><br>Valid Values:<br>• ProcessGroup<br>• FileExists |
| file_exists_file_name | When pre_req_type is set to 'FileExists' This is the fully qualified path name of the file that must exist before the Process Group will be started. |
| pre_req_process_group | When pre_req_type is set to 'ProcessGroup' This is the name of the Process Group that must be completed successfully before the Process Group will be started. |
| short_description | This column contains a short description of the Process Group that will be printed onto the reports for the batch processing. It is intended to assist the support personell understand the batches a little more easily. |
| long_description | This column contains a long description of the Process Group that will be printed onto the reports for the batch processing. It is intended to assist the support personell understand the batches a little more easily. |

#### *12.12.2.4.The Commands Table*

So, up until now you have not read anything about actually executing Processes (commands) in the scheduler. It has all been about Batches and Process Groups. Of course, at some point in the documentation of the Scheduler we must actually document how to run a Process (command).

This is the commands table.

```
create table dbo.ctl_commands
    (  pk_process_batch_name        varchar (255)     not null    default 'unknown'
      ,pk_process_group_name        varchar (255)     not null    default 'unknown'
      ,pk_process_step_number       integer           not null    default 0
      ,process_command_name         varchar (255)     not null    default 'unknown'
      ,process_program_name         varchar (255)     not null    default 'unknown'
      ,process_program_parms        varchar (4000)    not null    default 'unknown'
      ,process_program_name_fail    varchar (255)     not null    default 'unknown'
      ,process_program_parms_fail   varchar (4000)    not null    default 'unknown'
      ,short_description            varchar (4000)    not null    default 'unknown'
      ,long_description             varchar (4000)    not null    default 'unknown'

/* These three fields added to enable conditional running of commands.... */

      ,cmd_pre_req_flag             varchar (1)       not null    default 'N'
      ,cmd_pre_req_type             varchar (50)      not null    default 'unknown'
      ,cmd_file_exists_file_name    varchar (4000)    not null    default 'unknown'
)
;
```

The following table explains the values of each column and the functions available.

| Column Name | Function |
|---|---|
| pk_process_batch_name | This is the batch name assigned for the specific batch. It is the same batch name as occurs on the ctl_batch_pre_req table. A Process Group is placed into a batch by using the same name that defines the batch.<br><br>Example: batch001.<br><br>Default: 'unknown' |
| pk_process_group_name | This is the name of the Process Group. The name of the process group links this Process to the Process Group.<br><br>Example: group001, group002, group003 etc. |
| pk_process_step_number | This is the number of the Process in the Process Group. Processes are executed sequentially in the Process Group and the return code from each of the processes must be zero for the Process Group to continue processing.<br><br>Zero happens to be the standard return code of SeETL[RT]. |
| process_command_name | This is a character name that the user can give to a command. This name can be more descriptive than the program name itself. For example CTLU001.exe is the Data Transfer Utility. So the user might want to put DTU in the process_command_name as a reminder as to what process is actually running. |
| process_program_name | This is the program name that will be submitted to the 'system' command inside windows/unix.<br><br>**(2.1 Enhancement)** Previous versions required that this command be visible in the path of the userid running SeETL[RT]. Typically the command was in a file visible to the operating system and $PATH. In version 2.1 this approach can still be used. However, we have introduced a new table called ctl_process_commands. This table contains commands that can also be submitted to the operating system. The |

| | |
|---|---|
| | table can be used to eliminate the need to place SeETL[RT] commands into files visible to the $PATH. See below for a full description of the ctl_process_commands table.<br><br>The program name can actually be any program name that can be submitted at the command line of windows/unix. Thus the Batch Processing Scheduling Utility can be used to schedule and run any commands on windows and unix. (A handy utility indeed!)<br><br>Example.<br>To run the Data Transfer Utility on windows this field would contain CTLU001.exe. |
| process_program_parms | This field contains the parameters to pass to the program. In the case of SeETL[RT] this is the set of parameters needed to run whichever program that is being submitted using the process_program_name.<br><br>We have not included an example here as there are plenty of other examples in this document. |
| process_program_name_fail | If the command in process_program_name fails this is the program name that will be submitted to the 'system' command inside windows/unix to perform some kind of 'failed' command processing. The program name must be found in the path of the operating system. The scheduler does not allow for you to specify the path of the executable program. |
| process_program_parms_fail | This field contains the parameters to pass to the program that is run on the failure of the program specified in process_program_name. |
| short_description | This column contains a short description of the command that will be printed onto the reports for the batch processing. It is intended to assist the support personell understand the batches a little more easily. |
| long_description | This column contains a long description of the command that will be printed onto the reports for the batch processing. It is intended to assist the support personell understand the batches a little more easily. |
| cmd_pre_req_flag | This flag is set to 'Y' when the command has a pre-requisite file associated with it. If this flag is not set to 'Y' then the remainder of the line is ignored. |
| cmd_pre_req_type | This field is set to 'FileExists' or 'FileNotExists' depending on whether the user wants to test for the existence or non-existence of a file to determine if the command should be executed. |
| cmd_file_exists_file_name | This is the name of the file that will be tested for existence or non-existence when this pre-requisite is set. |

So, in this brief introduction you can see that a Process (command) is part of a Process Group and the Process Group is part of a Batch. This is how the hierarchy of the processes are constructed.

#### *12.12.2.5.The Process Commands Table*

In Release 2.1 we introduced the ctl_process_commands table. Up until Release 2.1 the user had to place the commands that would be executed into files on the operating system and then make those commands visible in the path of the userid that would execute the Batch.

As a part of our move to place aas much information as possible into the 'Mapping Spreadsheet' we have enhanced the Scheduler to optionally read the commands to be executed from the ctl_process_commands table.

When the Scheduler attempts to run a command defined in the ctl_commands table (above) it will first read this ctl_process_commands table to determine if the command exists in this table. It uses the process_program_name from the ctl_commands table to search the process_name in the ctl_process_commands table. If these two fields are the same the process_command in the ctl_process_commands table will be submitted to the operating system.

If the command does not exist in the ctl_process_commands table then the command and the parameters will be passed to the operating system unchanged.

If the command does exist in the ctl_process_commands table then the Scheduler will

1. Read the command from the ctl_process_commands table.
2. Perform the defined parameter substitution in the command.
3. Send the finished command to the operating system for execution.

In the SeETL[RT] Batch Processing Utility the parameter character is defined to be '?' and not '%' or '$' as is used in windows and unix based systems.

The reason is that it is also possible that the user will want to be able to pass windows or unix based parameters as well as SeETL[RT] based parameters to a command.

The maximum number of parameters that can be passed to a command is 9 (nine). They are specified as ?1. ?2. ?3 through to ?9. We have not allowed the use of ?0 as we felt that would be just too confusing.

The one parameter can be used in the command in many places. For example ?1 might be used in two or three places in the command itself.

There is currently very little 'error checking' around the use of parameters so we recommend that users are careful to ensure that the commands and parameters are correct before placing them into the Mapping Spreadsheet. For example ?A will not cause any substitution to take place. A substitution will only take place for ?1-?9.

This is the process commands table.

```
create table dbo.ctl_process_commands
    (  pk_int_key             integer                 not null primary key
    ,  process_name           varchar        (256)    not null default 'unknown'
    ,  process_command        varchar        (4000)   not null default 'unknown'
      ,short_description      varchar        (4000)   not null default 'unknown'
      ,long_description       varchar        (4000)   not null default 'unknown'
    )
;
```

The following table explains the values of each column and the functions available.

| Column Name | Function |
|---|---|
| pk_int_key | This is simply a sequential number which can be used as a primary key if desired. |
| process_name | This is the name of the Process that is entered into the ctl_commands table. For example this command might be 'RunType1Dimension' or 'RunFactTable1' or some such other command that is short hand for the longer commands that have typically been used. |
| process_command | This is the full command including parameter place holders that have been placed in files in previous versions.<br><br>The typical example is 'RunType1Dimension'.<br><br>In this field the user could place:<br><br>`CTLDM01.exe`<br>`DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=sa;`<br>`PWD=password;DATABASE=SEETL3000`<br>`InCatalogName=SEETL3000`<br>`InSchemaName=dbo`<br>`InTableName=?1`<br>`DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=sa`<br>`;PWD=password;DATABASE=SEETL3000 OutCatalogName=SEETL3000`<br>`OutSchemaName=dbo`<br>`OutTableName=?1`<br>`ErrorMessageOutput=TargetDatabase`<br>`DebugLevel=0`<br>`Audit=Yes`<br>`InputTableOrFileCanBeEmpty=Yes`<br><br>The user could then place into the ctl_commands table a call such as<br>process_program_name = RunType1Dimension<br>process_program_parms = customer_dim<br><br>This would then retrieve the process_command described above and substitute the ?1 parameter place holders with 'customer_dim' and then submit the final command directly to the operating system. |
| short_description | This column contains a short description of the process command that will be printed onto the reports for the batch processing. It is intended to assist the support personell understand the batches a little more easily. |
| long_description | This column contains a long description of the process command that will be printed onto the reports for the batch processing. It is intended to assist the support personell understand the batches a little more easily. |

This data can be entered directly into this table or, more typically, entered into the 'SeETL$^{DT}$ Spreadsheet' to be loaded into the table using SeETL$^{DT}$.

#### *12.12.2.6.The Batch Run Log Table*

Batches are monitored using the batch run log table.

```
create table dbo.ctl_batch_run_log
    (  pk_batch_number        integer           not null    default 0
      ,pk_process_batch_name  varchar     (255) not null    default 'unknown'
      ,pk_started_tstamp      datetime          not null
      ,ended_tstamp           datetime
      ,end_status             varchar     (20)  not null    default 'unknown'
    )
;
```

| Column Name | Function |
|---|---|
| pk_batch_number | This is the number of the batch from the ctl_batch_control table. It identifies the 'batch' that this particular batch was run under. It is important to understand that there are two meanings for a batch.<br><br>The first meaning is that a batch is a set of processes that can be run.<br><br>The second meaning is that a batch is the instance of that set of processes actually being run.<br><br>When processing is performed on a repeated basis the scheduler checks to see if the repetitive processing has already been run in the current 'batch'. If it has been run successfully it will not be started again. This way the daily processing of the batch cannot be accidentally re-run. |
| pk_process_batch_name | This is the batch name assigned for the specific batch. It is the same batch name as occurs on the ctl_batch_pre_req table. A Process Group is placed into a batch by using the same name that defines the batch.<br><br>Example: batch001.<br><br>Default: 'unknown' |
| pk_started_tstamp | This is the start timestamp of the batch. |
| ended_tstamp | This is the end timestamp of the batch. |
| end_status | This is the end status of the batch. If can be 'Running', 'Successful' or 'Abended'.<br><br>Once a batch is Abended it can only be re-started by the user going into this table and changing the end_status from 'Abended' to 'Restart'. When the end_status is set to 'Restart' the scheduler will re-submit the Process Groups that make up the batch and the Process Groups will restart at the point of the abend. |

#### *12.12.2.7.The Process Group Run Log Table*

Process Groups are monitored using the process group run log table.

```
create table dbo.ctl_proc_grp_run_log
    (  pk_batch_number              integer          not null    default 0
      ,pk_process_batch_name        varchar (255)    not null    default 'unknown'
      ,pk_process_group_name        varchar (255)    not null    default 'unknown'
      ,started_tstamp               datetime
      ,ended_tstamp                 datetime
      ,end_status                   varchar (20)     not null    default 'unknown'
    )
;
```

| Column Name | Function |
|---|---|
| pk_batch_number | This is the number of the batch from the ctl_batch_control table. It identifies the 'batch' that this particular batch was run under. It is important to understand that there are two meanings for a batch. |
| pk_process_batch_name | This is the batch name assigned for the specific batch. It is the same batch name as occurs on the ctl_batch_pre_req table. A Process Group is placed into a batch by using the same name that defines the batch.<br><br>Example: batch001.<br><br>Default: 'unknown' |
| pk_process_group_name | This is the name of the Process Group who's activity is being logged by this record. |
| started_tstamp | This is the start timestamp of the Process Group.<br>Note. The pk_ has been dropped from the beginding of this field name in Release 2.1. |
| ended_tstamp | This is the end timestamp of the Process Group. |
| end_status | This is the end status of the Process Group. If can be 'Running', 'Successful' or 'Abended'. The user should never update this field. |

#### *12.12.2.8.The Process Run Log Table*

Processes are monitored using the process run log table.

```
create table dbo.ctl_proc_run_log
    (  pk_batch_number              integer           not null    default 0
      ,pk_process_batch_name        varchar (255)     not null    default 'unknown'
      ,pk_process_group_name        varchar (255)     not null    default 'unknown'
      ,pk_process_step_number       integer           not null    default 0
      ,process_command_name         varchar (255)     not null    default 'unknown'
      ,process_program_name         varchar (255)     not null    default 'unknown'
      ,process_program_parms        varchar (4000)    not null    default 'unknown'
      ,started_tstamp               datetime
      ,ended_tstamp                 datetime
      ,end_status                   varchar (20)      not null    default 'unknown'
    )
;
```

| Column Name | Function |
|---|---|
| pk_batch_number | This is the pk_batch_number from the ctl_commands table. |
| pk_process_batch_name | This is the pk_process_batch_name from the ctl_commands table. |
| pk_process_group_name | This is the pk_process_group_name from the ctl_commands table. |
| pk_process_step_number | This is the pk_process_step_number from the ctl_commands table. |
| process_command_name | This is the process_command_name from the ctl_commands table. |
| process_program_name | This is the process_program_name from the ctl_commands table. |
| process_program_parms | This is the process_program_parms from the ctl_commands table. |
| pk_started_tstamp | This is the start timestamp of the Process Group. |
| ended_tstamp | This is the end timestamp of the Process Group. |
| end_status | This is the end status of the Process Group. If can be ‘Running’, ‘Successful’ or ‘Abended’. The user should never update this field. |

### 12.12.3.What Does the Batch Processing Scheduling Utility do?

As the name implies the Batch Processing Scheduling Utility provides the ability to schedule batches of processes.

It allows a batch to be start based on any number of pre-requisites being met. The current list is:

- FileExists
- Daily
- Weekly
- Monthly
- OneTime

A batch can consist of one or more Process Groups. Process Groups can themselves have pre-requisites of:

- FileExists
- Another Process Group

A Process Group can then consist of a series of Processes which run sequentially.

The Batch Processing Scheduling Utility has been built so that it can maximise the usage of parallel processing hardware by running many independent process groups at the same time. However, in data warehousing there are usually some very specific processing dependencies.

Data warehousing batch processing generally acts a little differently to batch processing of a multitude of large systems in that there are some specific points which must be reached before any other processing can continue reliably. For example, in general, all dimension tables must be updated prior to starting any fact table process.

Let us consider a simple example.

Let's say the data warehouse batch has a nightly run of the following:

1. 100 tables to be put into the staging area
2. 30 dimension tables to be generated from around 80 staging tables
3. 20 fact tables to be processed from the other 20 staging tables.

Let's also say that each staging table, dimension table and fact table have one process that runs to produce the required result. Though this is a bit simplistic the reader will get the idea.

Obviously it is desirable, but not absolutely necessary that all staging tables are loaded before any dimension tables. It is necessary to have the staging tables that are required for a specific dimension table loaded before the dimension table can be updated.

This batch can be built a number of different ways with differing processing elapsed times. The Batch Processing Scheduling Utility gives the user the ability to design the batch to get through the greatest amount of processing in the shortest possible time.

For example, the batch can be built as a single stream. The user can define one batch to start at a specific point in time. That batch could contain one process group and that process group could contain all the processes in order.

Batch001
    Group001
        Process001
        Process002
        …
        Process150

Clearly such a schedule will usually only only use 1 processor throughout so it will be very slow if you have a 4 processor machine.

Let's say we have a 4 processor windows machine and we want to do as much as possible as fast as possible. In this case we might construct a batch as follows:

Batch001-Process All Loading Requirements

Group001
- Process001
- Process002
- …
- Process025

Group002
- Process026
- Process027
- …
- Process050

Group003
- Process051
- Process052
- …
- Process075

Group004
- Process076
- Process077
- …
- Process100

Group005 Depending on Group001/002/003/004 ← Semaphore Process Group
- Process100A (which does nothing)

Group006 Depending on Group005
- Process101
- Process102
- …
- Process108

Group007 Depending on Group005
- Process109
- Process110
- …
- Process116

Group008 Depending on Group005
- Process117
- Process118
- …
- Process124

Group009 Depending on Group005
- Process125
- Process126
- …
- Process130

Group010 Depending on Group006/007/008/009 ← Semaphore Process Group
- Process130A (which does nothing)

Group011 Depending on Group010
- Process131
- Process132
- …
- Process135

Group012 Depending on Group010
- Process136
- Process137
- …
- Process140

Group013 Depending on Group010
- Process141
- Process142

```
            …
            Process145
      Group014      Depending on Group010
            Process146
            Process147
            …
            Process150
```

In the above example, each set of Process Groups is set up to run 4 Process Groups at the same time. On windows, this would keep the processors at a reasonably high utilisation assuming that the workloads in each Process Group at each stage of processing were relatively similar in processing times.

The reader should be aware that one of the current 'limitations' of the Batch Processing Scheduling Utility is that the process that keeps watch on the Process Groups requires 4 connections to the database via ODBC to be open at all times. Also, when a batch starts a Process Group all Process Groups in the batch are started as 'set and forget'. They then take care of themselves in watching for their pre-requisite process groups to complete or files to be created. This can lead to a lot of open connections which you may want to avoid.

So in the above example the process that takes care of Process Groups will be started 14 times and each invocation will open 4 connections to the database. That is 56 open connections. This is before any process actually starts!!!

Although most databases will support this, it is probably better for you to be aware of this overhead in the current version and to design batches and process groups with this limitation in mind.

The process that looks after batches requires 5 open connections to the database but there is only ever one copy of the scheduler running on a single machine.

The above batch could be re-written just as effectively as follows:

```
Batch001-Process Staging Area
      Group001
            Process001
            Process002
            …
            Process025
      Group002
            Process026
            Process027
            …
            Process050
      Group003
            Process051
            Process052
            …
            Process075
      Group004
            Process076
            Process077
            …
            Process100
      Group005      Depending on Group001/002/003/004    ← Semaphore Process Group
            Process100A (create a semaphore file to mark staging area complete.)

Batch002-Process Dimension Tables (Depending on semaphore file)
      Group006
            Process101
            Process102
            …
            Process108
      Group007
            Process109
            Process110
```

…
Process116
Group008
Process117
Process118
…
Process124
Group009
Process125
Process126
…
Process130
Group010 Depending on Group006/007/008/009 ← Semaphore Process Group
Process130A (create a semaphore file to mark dimension tables as complete)

Batch003-Process Fact Tables (Depending on semaphore file)
Group011
Process131
Process132
…
Process135
Group012
Process136
Process137
…
Process140
Group013
Process141
Process142
…
Process145
Group014
Process146
Process147
…
Process150
Group015 Depending on Group011/012/013/014 ← Semaphore Process Group
Process150A (delete semaphore files for staging and dimensions)

This set of batches and Process Groups will perform the same processing as the previous version only it will have just 5 instances of the process to manage Process Groups running at any one time which will only require 20 open connections, not 56 open connections.

Of course, if you know which staging area tables must be loaded before which dimension tables it is always possible to move the dimension table processing into the first batch so that there is no 'waiting' for the staging area to be complete before the dimension table processing can be started.

On windows it is recommended to be running only the same number of concurrent processes as there are processors in the machine. On unix, it is expected that the user can run twice as many processes as there are processors in the machine.

Given the flexibility of building the hierarchy of Batches, Process Groups and Processes the user may want to experiment with the best way in which to construct batches for processing loads. The user should just keep in mind not to have too many process groups running and waiting for their pre-requisites to be completed because they will be taking 4 connections each to the database.

### 12.12.4.How Does the Batch Processing Scheduling Utility Work?

In order for the user to be able to make better decisions on how to set up the Batches and Process Groups this section is included to explain how the Batch Processing Scheduling Utility works.

The Batch Processing Scheduling Utility comprises two parts:

1. CTLU008 – The Batch Scheduler
2. CTLU009 – The Process Group Manager

The following process takes place:

1. The Batch Scheduler (CTLU009) runs all the time. There should be only one instance of this program running at any given time.

2. It sleeps for a parameter specified time and then wakes up.

3. When it wakes up it checks the set of pre-requisites for all batches.

4. If a batch is found to pass it's pre-requisites the Batch Scheduler reads the dw commands table to find all the Process Groups in the batch.

5. The Batch Scheduler then starts the Process Group Manager, one time for each Process Group in the batch. It passes the Batch and the Process Group to the Process Group Manager.

6. The Process Group Manager (for each instance that is running) then checks the pre-requisites for the Process Group. It sleeps for a parameter specified period before waking up again and checking it's pre-requisites again.

7. When the Process Group Manager detects that the pre-requisites for the Process Group have been met then it will start running each of the processes in the Process Group. It will run the processes in order and it will wait for each process to complete prior to moving on to the next process.

So, from this description, you can see that it is slightly less load on the system to design batches which are dependent on each other using files rather than creating large numbers of process groups that are dependent on each other that are all in the one batch.

It is recommended that the most efficient way to build batches is to group together sets of workloads to be processed down a single processor that will require roughly equal elapsed times. These groups of processors should be placed into batches with the final Process Group being dependent on all the other Process Groups in the batch. When all the Process Groups in the batch complete the final Process Group should do nothing but copy/rename a semaphore file to release the next batch of processing to the machine.

This will result is some small loss of processing time when the Process Groups finish in an uneven order. However, all Process Groups and Processes are fully logged in the run logs. It is recommended that the processing run logs are reviewed on a regular basis to ensure that the Process Groups and Batches are running as fast as is desired and that there have been no large changes in the processing times of specific Process Groups.

## 12.12.5.Examples of Invoking the Batch Processing Scheduling Utility

The Batch Processing Scheduling Utility is one of the easiest of all utilities to run. This is because it takes most of it's input from the tables described in this document. The user really just needs to start the program and tell it where to send it's error messages (recommended to be the TargetDatabase) and whether to be debugging the program.

CTLU008.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=not_used
ErrorMessageOutput= TargetDatabase
DebugLevel=0
Audit=No
KillFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\ctlu008.kill.yes
SchedulerCanStartBatch=Yes
WaitFileName=C:\IBISoftware\SeETL\DesignTime\3.0.01\TestData\CTLU008.wait.yes

Note that the Batch Processing Scheduling Utility runs endlessly and so a 'kill' file has been added. The name of the parameter is KillFileName. When this file is seen to exist by the Batch Processing Scheduling Utility it will delete the file and stop processing. This is just an easy way of stopping the endless processing of the scheduler rather than having to find the process and stop it using some other means such as a system command or the windows task manager.

**3.0.00 Enhancement**

In SeETL We made significant effort to stop accidental double processing. We have utilities called Batch Maintenance Utilities to maintain a semaphore around the batch so that a deliberate act is required to open up the batch for processing.

Until now CTLBM01 must have been started by some process outside the Scheduler to allow the scheduler to start processing.

One of our large clients requested that we add this capability to the scheduler. That is, allow the scheduler to start the batch. We have included this in SeETL RT 3.0.00. This parameter tells the scheduler that the user has specifically decided that the scheduler can start batch processing based on the WaitFileName parameter that must also be supplied.

When the SchedulerCanStartBatch parmeter is set to Yes the scheduler will do the following.

1. Determine of the ctl_batch_control table indicates a batch is already running.
    a. If Yes it will process the batch normally.
    b. If No it will wait untl the file named in the WaitFileName parameter is present. When it is present it will insert a new row into the ctl_batch_control table to open up the processing of a batch and delete the file named in the WaitFileName parameter.

## 12.12.6.What is the Process Group Manager (CTLU009)

The Process Group Manager is program CTLU009. As far as the user is concerned there is no need to know anything about this program. It is called by the Batch Processing Scheduling Utility and is never invoked directly by the user. However, we are something of believers in documentation and so we have included this small section on the Process Group Manager.

The Process Group Manager is the program that manages the execution of a single Process Group. The Batch Processing Scheduling Utility reads through all the commands scheduled in the batch and all the process groups that are scheduled in the batch. It then calls CTLU009 to start the process groups. The relevant code in the Batch Processing Scheduling Utility is as follows:

```
////////////////////////////////////////////////////////////////////////////////
// Generate an sql statement to select all the process groups for the batch.     //
////////////////////////////////////////////////////////////////////////////////

strcpy (ctl_commands_batch_proc_grp_asn_SQL_Statement,
            "SELECT  distinct pk_process_batch_name , pk_process_group_name FROM ") ;
strcat (ctl_commands_batch_proc_grp_asn_SQL_Statement,
            ctl_commands_batch_proc_grp_asn_Fully_Qualified_Name ) ;
strcat (ctl_commands_batch_proc_grp_asn_SQL_Statement,
            " where pk_process_batch_name = '" ) ;
strcat (ctl_commands_batch_proc_grp_asn_SQL_Statement,
            ws_bpr_pk_process_batch_name ) ;
strcat (ctl_commands_batch_proc_grp_asn_SQL_Statement, "' order by
            pk_process_batch_name , pk_process_group_name " ) ;

ctl_commands_batch_proc_grp_asn_Fully_Qualified_Name = ctl_commands table.
```

The Process Group name is retrieved from the table as follows:

```
strcpy(ws_cbpga_pk_process_group_name           ,
      ptr_ctl_commands_batch_proc_grp_asn->ptr_table_field_desc_array[1].field_data)
;

strcpy(ws_CTLU009_parameter,"") ;
strcat(ws_CTLU009_parameter," DBConnectionOutParameter=") ;
strcat(ws_CTLU009_parameter,  DBConnectionOutParameter) ;
strcat(ws_CTLU009_parameter," OutCatalogName=") ;
strcat(ws_CTLU009_parameter,  OutCatalogName) ;
strcat(ws_CTLU009_parameter," OutSchemaName=") ;
strcat(ws_CTLU009_parameter,  OutSchemaName) ;
strcat(ws_CTLU009_parameter," OutTableName=") ;
strcat(ws_CTLU009_parameter,  OutTableName) ;
strcat(ws_CTLU009_parameter," TargetDataBase=") ;
strcat(ws_CTLU009_parameter,  TargetDataBase) ;

strcat(ws_CTLU009_parameter," ProcessBatchName=") ;
strcat(ws_CTLU009_parameter,  ws_bpr_pk_process_batch_name) ;
strcat(ws_CTLU009_parameter," ProcessGroupName=") ;
strcat(ws_CTLU009_parameter,  ws_cbpga_pk_process_group_name) ;
strcat(ws_CTLU009_parameter," ErrorMessageOutput=TargetDatabase") ;
```

Note that for CTLU009 the error message output must be sent to the target database. Since this process is run as a background process it is not sensible or useful to write the error messages anywhere else other than to the target database.

Once the parameters to CTLU009 are properly created CTLU009 is started by the spawn command as follows:

```
ws_spawn_return_code =  spawnlp(P_NOWAIT,ws_CTLU009_program_name,
                        ws_CTLU009_program_name,ws_CTLU009_parameter,NULL) ;
```

In this way CTLU009 is started with the Process Group Name as a parameter. The user will immediately see that if there are N Process Groups in a batch all Process Groups will be started by this code. Spawnlp is the command to spawn a process where the command that will be issued is in the path. The P_NOWAIT parameter tells the spawn command to start the command and to then return to the calling program. It does not wait for the spawned process to finish. In this case the ws_spawn_return_code will be the process id of the called process.

Once CTLU009 is started it waits for any pre-requisites, which could be the completion of another Process Group or the existence of a file, and then it starts to find the commands that are in the Process Group using the following call.

The following call finds all the commands in the Process Group Name passed to CTLU009.

```
strcpy (Select_ctl_commands_table, "SELECT pk_process_step_number,
        process_command_name,  process_program_name, process_program_parms FROM ") ;
strcat (Select_ctl_commands_table, ctl_commands_table_Fully_Qualified_Name ) ;
strcat (Select_ctl_commands_table, " where pk_process_batch_name ='" ) ;
strcat (Select_ctl_commands_table, ProcessBatchName ) ;
strcat (Select_ctl_commands_table, "' and pk_process_group_name ='") ;
strcat (Select_ctl_commands_table, ProcessGroupName ) ;
strcat (Select_ctl_commands_table, "' order by pk_process_step_number " ) ;
```

The data is read from the ctl_commands table and placed into working storage as follows:

```
strcpy(ws_pk_process_step_number    ,
      ptr_ctl_commands_table->ptr_table_field_desc_array[0].field_data) ;
strcpy(ws_process_command_name            ,
      ptr_ctl_commands_table->ptr_table_field_desc_array[1].field_data) ;
strcpy(ws_process_program_name            ,
      ptr_ctl_commands_table->ptr_table_field_desc_array[2].field_data) ;
strcpy(ws_process_program_parms           ,
      ptr_ctl_commands_table->ptr_table_field_desc_array[3].field_data) ;
```

It then fetches the system start time, prepares the command to execute, executes the command and then fetches the system start time again so it can log the start and stop time for each process. The code to perform this processing is as follows:

```
////////////////////////////////////////////////////////////////////////////
// Prepare the command to execute.                                        //
////////////////////////////////////////////////////////////////////////////

strcpy(ws_command_to_execute,ws_process_program_name) ;
strcat(ws_command_to_execute," ") ;
strcat(ws_command_to_execute,ws_process_program_parms) ;

////////////////////////////////////////////////////////////////////////////
// Get the system date time before the start of the command.              //
////////////////////////////////////////////////////////////////////////////

RU004_GetSystemDateTime (ws_TargetDataBase , ws_process_start_timestamp) ;

return_code = system( ws_command_to_execute );

////////////////////////////////////////////////////////////////////////////
// Get the system date time at the end of the command.                    //
////////////////////////////////////////////////////////////////////////////

RU004_GetSystemDateTime (ws_TargetDataBase , ws_process_end_timestamp) ;
```

Inside CTL009 each row of the ctl_commands tables for this Process Group is read in the order of the `pk_process_step_number` field. The command is then executed but CTLU009, unlike CTLU008 waits for the result of the call to the program represented by ctl_commands.process_program_name. In this way, dependencies between programs in a single Process Group are established by simply running them in the order that the dependencies exit. It is not possible to run Processes in a Process Group out of order.

Because the 'system' command is used to execute the ctl_commands.process_program_name with the ctl_commands. process_program_parms it is clear that any command that can be executed at the system prompt of the windows or unix prompt can be executed by CTLU009.

## *12.13. DataStage Job Submission Utility – No Parameters*

### 12.13.1.Why Have the DataStage Job Submission Utility?

We do as lot of project work with DataStage and Informatica. On a recent project we had to read a large number of telco CDR files coming from the mediation system in the binary unformatted switch format. Because we did not want to buy another license of one of the vendors switch decoders we decided to just build the switch decoders in C++. We also wanted to write as little C++ as possible and have as much code in DataStage as possible. So the Switch Decoding Software simply decoded the switches and put the data into a format that was readable by DataStage. This was either fixed format ascii or delimited ascii files.

We were going to write a separate routine to process files arriving in folders and another routine to read the decided files and process them in DataStage. However three 'moving parts' is always more unreliable than one 'moving part' so we decided to try and find out if we could extend the Switch Decoding Software to so that it would also read the input directories and submit the DataStage jobs.

It turned out to be quite easy to do and the solution to this problem for the customer is simply one C++ program that can scan through any input directory looking for binary unformatted switch files to decode, decode them, and then start the DataStage job to load the decoded switch file into the staging area.

We also included the ability to put the process to sleep as well as to kill the process through the existence of particular files in the directory where the switch records were located.

However, particular to the DataStage Job Submission Utility, we had to learn how to run DataStage jobs from inside C++ to make this processing work. This activity turned out to be easier than we thought so we decided to make a free utility out of it and offer it to the DataStage community. Note that the 'free' version is only available on windows2000. However, a windows 2000 machine can be used to run the DataStage Batch Schedule on any unix machine.

(The unix versions of all our utilities are 'fee' products because we supply the source code for unix customers. We don't have unix machines available to be able to support the unix versions of code so customers must support unix platforms themselves.)

The other main reason for having the DataStage Job Submission Utility is the issue of controlling parallelization and dependencies in a large and complex DataStage Batch Schedule. DataStage relies on the batch control language in order to schedule and control parallelization and dependencies in batches. When there is a need to re-start a batch there is manual work involved in the restart process. Manual changes in a restart process is always fraught with danger. Mistakes are made and databases are corrupted or damaged. It is simply what happens when people need to make changes to complex things in the middle of the night.

The DataStage Job Submission Utility can be used in conjunction with the Batch Processing Scheduling Utility to provide an extremely robust mechanism for scheduling DataStage jobs. Certainly a far more robust and sophisticated Batch Scheduling Solution than the DataStage Scheduler.

### 12.13.2.What Does the DataStage Job Submission Utility Do?

The DataStage Job Submission Utility is very simple and only performs one function. It runs a DataStage Job. You pass it the following parameters using the normal command line interface of all SeETL[RT] Utilities and it will execute the job.

- `DataStageServerName`
- `DataStageProjectName`
- `DataStageUserName`
- `DataStageUserPassword`
- `DataStageJobName`

If the job executes successfully the DataStage Job Submission Utility will return a zero return code.

### 12.13.3.How Does the DataStage Job Submission Utility Work?

First of all, there is extensive documentation supplied with DataStage as to how to call DataStage jobs from C++. It is in the Server Job User Guide. The DataStage Job Submission Utility simply uses this API in order to call jobs.

The actual code is as follows:

```
////////////////////////////////////////////////////////////////////////////////
// get a handle for the project.                                              //
////////////////////////////////////////////////////////////////////////////////

      ws_hProject = DSOpenProject(DataStageProjectName) ;

////////////////////////////////////////////////////////////////////////////////
// If we cannot get a handle for the project then issue a severe error message   //
// and stop processing.                                                       //
////////////////////////////////////////////////////////////////////////////////

      IF (ws_hProject EQUALS_ NULL) THEN
        BEGIN_
            <error processing>
        END_


////////////////////////////////////////////////////////////////////////////////
// Open the job to get ready to lock it and run it.                           //
////////////////////////////////////////////////////////////////////////////////

    ws_hJob = DSOpenJob(ws_hProject, DataStageJobName) ;

////////////////////////////////////////////////////////////////////////////////
// If the job cannot be opened then issue an error message and stop.          //
////////////////////////////////////////////////////////////////////////////////

      IF (ws_hJob EQUALS_ NULL) THEN
        BEGIN_
            <error processing>
        END_


////////////////////////////////////////////////////////////////////////////////
// Lock the job to get ready to run it.                                       //
////////////////////////////////////////////////////////////////////////////////

    ws_status = DSLockJob(ws_hJob) ;

////////////////////////////////////////////////////////////////////////////////
// If the job cannot be locked then issue an error message and stop.          //
////////////////////////////////////////////////////////////////////////////////

      IF (ws_status NOT_EQUALS_ DSJE_NOERROR) THEN
        BEGIN_
            <error processing>
        END_

////////////////////////////////////////////////////////////////////////////////
// Run the job.                                                               //
////////////////////////////////////////////////////////////////////////////////

      ws_status = DSRunJob(ws_hJob, DSJ_RUNNORMAL);

////////////////////////////////////////////////////////////////////////////////
// If the job cannot be run properly then issue an error message and stop.    //
```

```
////////////////////////////////////////////////////////////////////////////////

      IF (ws_status != DSJE_NOERROR) THEN
        BEGIN_
            <error processing>
        END_

////////////////////////////////////////////////////////////////////////////////
// Wait for it to finish.                                                     //
////////////////////////////////////////////////////////////////////////////////

      ws_status = DSWaitForJob(ws_hJob);

////////////////////////////////////////////////////////////////////////////////
// If the wait for the job to finish failes then issue an error message and stop.//
////////////////////////////////////////////////////////////////////////////////

      IF (ws_status != DSJE_NOERROR) THEN
        BEGIN_
            <error processing>
        END_

////////////////////////////////////////////////////////////////////////////////
// If we got here the job ran ok so we need to unlock it and close it.        //
// We also need to close the project.                                         //
////////////////////////////////////////////////////////////////////////////////

      DSUnlockJob(ws_hJob);
      DSCloseJob(ws_hJob);
      DSCloseProject(ws_hProject);
```

### 12.13.4.Integration with the Batch Processing Scheduling Utility

The DataStage Job Submission Utility itself is nothing that special. It is very simple and only performs one function. It runs a DataStage Job. However, because it presents a command line interface that will run DataStage jobs it can be integrated into the Batch Processing Scheduling Utility.

There are a number of features in the Batch Processing Scheduling Utility that makes this integration with DataStage a very important and powerful tool.

When used with the DataStage Job Submission Utility the Batch Processing Scheduling Utility can:

- Start many DataStage jobs and job sequences at the same time and run them in parallel.
  In this way it is possible to run as many independent jobs at the same time as is necessary to maximize the use of the processors available on the machine. This has always been a problem with managing jobs using the DataStage control language.

- Manage the dependencies between streams of jobs so that they can all be bought to a successful conclusion prior to starting the next phase of processing. For example it is possible to run all jobs for the staging area before starting processing for the dimension table processing.

  In managing these relationships and dependencies between jobs the Batch Processing Scheduling Utility can ensure that certain processes are completed before others while still maximizing the use of the processors available. The user has complete control as to how to construct Batches and Process Groups of DataStage jobs.

- When a failure occurs the user simply requests to 'Restart' the batch and the Batch Processing Scheduling Utility will determine what jobs have completed successfully and which ones have not completed successfully. It will then Restart the jobs that have not yet completed successfully. This is something that DataStage has never been able to do. It has always required human intervention to restart a failed batch and it has required changes to the control language code that runs jobs to get the batch started again.

In short, the Batch Processing Scheduling Utility used in conjunction with the DataStage Job Submission Utility provides a more robust, reliable, easy to use interface than the DataStage control language itself and it also allows experienced DataStage users to construct Batches of DataStage jobs that will execute more quickly because of the increased level of control over the grouping together of jobs that the Batch Processing Scheduling Utility provides.

And this is all for free on windows 2000 even if DataStage is running on a unix platform.

### 12.13.5.Tables Required in the Target Database

None.

### 12.13.6.Examples of Invoking the DataStage Job Submission Utility

The DataStage Job Submission Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

```
CTLU010.exe
DBConnectionOutParameter=DSN=CTLOR01DB01;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE
=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=not_used
ErrorMessageOutput=TargetDatabase
DebugLevel=0
Audit=Yes
DataStageServerName=myDataStageServer
DataStageProjectName=myDataStageProject
DataStageUserName=myDataStageUserName
DataStageUserPassword=myDataStagePassword
DataStageJobName=myDataStageJobName
```

The DataStageServerName can be any value that is valid to enter in the normal DataStage Client. This includes such values as ‘localhost’, an ip address or a server name that is known to your local hosts file on your datastage client.

The job will succeed if DataStage only returns zero return codes. It will fail if DataStage signals any kind of error.

## *12.14. DataStage Job Submission Utility – With Parameters*

### 12.14.1.Why Have the DataStage Job Submission Utility – With Parameters?

One of the problems that must be solved when using DataStage is how the jobs will be parameterized so that they can be moved between environments and job processing can be controlled by the parameters.

The DataStage Job Submission Utility just allows a job to be started. The job itself must take care of parameters. This usually means a job needs to be written to start a job because the first job must set the parameters for the second job. That is, two jobs are required to be able to start individual jobs.

This seems something of a waste when testing DataStage jobs. So we wrote a utility to set the parameters of a DataStage job from the Data Warehouse database. In this environment we could just start a job on our remote DataStage server by just typing (or in fact recalling) a command on our command line on our desktops. Much faster and easier than going into director to start the job!!

### 12.14.2.What Does the DataStage Job Submission Utility Do?

The DataStage Job Submission Utility is very simple and only performs one function. It runs a DataStage Job. You pass it the following parameters using the normal command line interface of all SeETL[RT] Utilities and it will execute the job.

- `DataStageServerName`
- `DataStageProjectName`
- `DataStageUserName`
- `DataStageUserPassword`
- `DataStageJobName`
- `UseDataStageParameterTable`
- `DataStageParameterTable`

If the job executes successfully the DataStage Job Submission Utility will return a zero return code.

### 12.14.3.How Does the DataStage Job Submission Utility Work?

First of all, there is extensive documentation supplied with DataStage as to how to call DataStage jobs from C++. It is in the Server Job User Guide. The DataStage Job Submission Utility simply uses this API in order to call jobs. This program simply retrieves the parameter from the `DataStageParameterTable` and sets that parameter on the job before submitting it. We have not reproduced the code here because it is published on our web site.

### 12.14.4.Tables Required in the Target Database

- DataStageParameterTable

This table maintains the parameters for the jobs. It can be a view or a table. If you would like to be able to customise the parameters according to the specific jobs that are run you would add a column such as job_name or job_group and you would present a different view for the different jobs or job groups.

The program expects to see three columns as below for it's parameters and this is what the view must present.

```
create table dbo.ctl_datastage_parms
    (  pk_parm_number                        integer    not null  primary key
    ,  parm_name                          varchar (255)
    ,  parm_value                         varchar (4000)
    )
;
```

### 12.14.5.Examples of Invoking the DataStage Job Submission Utility

The DataStage Job Submission Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

```
CTLU010.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SE
ETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=not_used
ErrorMessageOutput=TargetDatabase
DebugLevel=0
Audit=Yes
DataStageServerName=myDataStageServer
DataStageProjectName=myDataStageProject
DataStageUserName=myDataStageUserName
DataStageUserPassword=myDataStagePassword
DataStageJobName=myDataStageJobName
UseDataStageParameterTable=Yes
DataStageParameterTable=ctl_datastage_parms
```

The DataStageServerName can be any value that is valid to enter in the normal DataStage Client. This includes such values as 'localhost', an ip address, or a server name that is known to your local hosts file on your datastage client.

The job will succeed if DataStage only returns zero return codes. It will fail if DataStage signals any kind of error.

## 12.15. Load Dimension Tables into Memory Maps Utility

A major improvement in scalability for SeETL[RT] is the ability to load dimension tables into memory maps and have the attribution processes only require one copy of dimension tables to look up no matter how many attribution processes are running at a given point in time.

The use of memory mapped IO introduces ***unlimited scalability*** to SeETL[RT]. The simplest way of thinking about it is that only one copy of lookup tables need be loaded into the memory of the operating system and then all attribution processes can view that one copy. This provides a massive reduction in memory requirements for large customers.

Please note.
For windows executables only licensed customers the Load Dimension Tables into Memory Maps Utility is licensed separately. It is not provided as a part of the standard SeETL[RT] product. It is only intended to be used by larger customers.

### 12.15.1. Why Have the Load Dimension Tables into Memory Maps Utility?

Pretty simple. It enables SeETL[RT] to scale endlessly with no memory restrictions/limitations on the files used for lookup processing. To process more transactions per day the user can simply acquire more CPUs and run more attribution processes on these CPUs. Memory mapped IO means there is no substantial increase in memory requirements and no loss of performance by adding more CPUs and running more attribution processes.

### 12.15.2. How Does the Load Dimension Tables into Memory Maps Utility Work?

The logic of how it works is simple. The code for how it works is more complex. For users who have a source code license it is suggested you view the source code to see the details of what is being done inside the Load Dimension Tables into Memory Maps Utility.

How the Load Dimension Tables into Memory Maps Utility works is as follows:

1. It reads the dim_table_load_control table to determine which dimension tables should be loaded into memory.

2. For each table it:
    a. Reads through the table counting the number of columns and the actual length of the string keys in the dim_char_ky_fld.

    b. Creates a file and allocates it to be the size required by all the keys and integers in the dimension table.

    c. Loads the dimension table into the memory mapped array which will, in turn, write the data into the memory mapped file.

3. Once all tables are loaded the Load Dimension Tables into Memory Maps Utility simply sits there and waits for it's 'kill file' to appear.

The Load Dimension Tables into Memory Maps Utility must remain resident as the 'provider' of memory mapped files during the processing of all records through the attribution process. In order to do this it has a parameter called 'KillFileName'.

The program will start and run through to the end of all required processing and then it will watch for the file 'KillFileName' to appear. By remaining resident all the attribution processes can use the memory mapped files it has created.

The Load Dimension Tables into Memory Maps Utility writes to a file called the WaitFileName to signal that it has completed the loading of all dimension tables into memory mapped arrays. Processing that is dependent on the fact that all the memory maps are resident in memory should be dependent on the existence of the file name passed in WaitFileName. The WaitFileName will only exist while all dimension tables are loaded into memory mapped arrays as the Load Dimension Tables into Memory Maps Utility will delete the file at the end of processing.

### 12.15.3.Tables Required in the Target Database

- dim_table_load_control table

### 12.15.4.Environment Variables Required in the Operating System

The Load Dimension Tables into Memory Maps Utility looks up the environment variable called SeETL[RT] TEMP to find the temporary directory for the memory mapped files to be placed in. Should this environment variable not be present in the operating system it will default to “D:\TEMP” on windows platforms.

Please note that the final slash is expected **NOT** to be in the directory name. It is added by the Load Dimension Tables into Memory Maps Utility.

### 12.15.5.Examples of Invoking the Load Dimension Tables into Memory Maps Utility

The Load Dimension Tables into Memory Maps Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

```
CTLU012.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=sa;PWD=password;
DATABASE=SEETL3000 InCatalogName=SEETL3000 InSchemaName=dbo  InTableName=not_used
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=sa;PWD=password;
DATABASE=PBDWPROD OutCatalogName=PBDWPROD OutSchemaName=dbo  OutTableName=not_used
ErrorMessageOutput=cerr
DebugLevel=0
Audit=No
KillFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLU012.kill.yes
WaitFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLU012.wait.yes
```

Notice that it is usually best to set both the input and output parameters to the data warehouse database. In fact this program uses the ‘In’ parameters because it is reading data ‘In’ from the data warehouse and is not reading from any of the staging area databases.

In the example above the program will look for the file D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLU012.kill.yes and it will end when it detects the existence of this file. So, to have the program run it is suggested that a file called D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLU012.kill.no is copied to D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLU012.kill.yes when all attribution processing has completed. This is a simple task to put into the schedule.

The user should be aware that once D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLU012.kill.yes has been detected the program will delete the file and end processing. In this way the user does not need to remember to delete the file before the next invocation of this program.

The parameter WaitFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLU012.wait.yes is the file that will be created by the Load Dimension Tables into Memory Maps Utility during the life of the program. Once all memory maps have been created and loaded this file will appear in the operating system.

Therefore, all attribution processing should detect the existence of this program. When the program detects the KillFileName it will also deleted the WaitFileName just before completing. Hence, if the attribution processes do not see the file WaitFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLU012.wait.yes in the operating system then the Load Dimension Tables into Memory Maps Utility is not running and has not loaded the memory maps.

The user should be aware that the schedule cannot be built such that attribution processing is dependent on the Load Dimension Tables into Memory Maps Utility itself because this program does not run to completion before the attribution processing starts.

## *12.16. MetaData Checking Utility*

SeETL[RT] is 'Typeless' and it does not check data types at run time. This is one of it's great features. If a data type can be converted to the target type by the ODBC driver it will be converted. The user does not need to perform any extra tasks to move that field. More specifically, the software does not issue 'MetaData Mismatch' warnings that need to be hunted down and removed like some of the vendor ETL tools.

A character string can be moved to an intger if it contains an integer. A character string can be moved to a date if it contains a date. And so on.

However, experience in using SeETL[RT] has shown that one of the more difficult 'bugs' to find is where a field of the incorrect data type/contents is sent to another field. This is particularly a problem because all of the inputs to SeETL[RT] are views and it is not easy to see the data types of the fields in the views. (See the MetaData Loading Utility next).

To help find these errors the MetaData Checking Utility inspects all source and target columns by name and issues messages where it detects that there might be a problem.

It has already proven to be a very useful utility in finding such things as right data truncation as well as accidental truncation of fields by using a taget column that is too short.

### 12.16.1. Why Have the MetaData Checking Utility?

Pretty simple. It saves time and effort debugging data movements that turn out to be mistakes

### 12.16.2. How Does the MetaData Checking Utility Work?

This utility is really simple. The user passes the parameters required for the the MetaData Checking Utility to connect to a source table and a target table. They can be in different databases or on different machines. These are the sources and targets that will be processed through other utilities like the Attribution Process or Dimension Processing.

The MetaData Checking tool connects to the source and the target and then it checks the data types of each field where the field name is found in both tables.

### 12.16.3. Tables Required in the Target Database

None.

## 12.16.4.Examples of Invoking the MetaData Checking Utility

The MetaData Checking Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

CTLU013.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=sa;PWD=password;
DATABASE= SEETL3000 InCatalogName= SEETL3000 InSchemaName=dbo InTableName=%1
DBConnectionOutParameter=DSN= SEETL3000;SERVER=PETERLAP;UID=sa;PWD=password;
DATABASE=PBDWPROD OutCatalogName=PBDWPROD OutSchemaName=dbo OutTableName=%1_dm
ErrorMessageOutput=cerr DebugLevel=0 Audit=No ReportSafeMetadataMismatch=No

In the above example the InTable nd OutTable are related by being called <tablename> and <tablename>_dm which is a check for a dimension processing program.

This command would be called by another command such as
```
checkmetadata dim_table_1
```

## *12.17. MetaData Printing Utility*

One of the real 'missing features' of most databases is that it is not easy to query their catalogs to determine the data types of columns in views. This is quite frustrating. SeETL[RT] makes extensive use of views and often times it would be very handy to just have a printout of the data types of the views. The MetaData Printing Utility prints the metadata for a single table to a single work file in the self describing file format used for other data. It is easy to read and has come in very handy.

### 12.17.1. What Does the Metadata Printing Utility Do?

The MetaData Printing Utility prints the metadata for a table to the workfile in the self describing file format supported for all other activities in SeETL[RT].

### 12.17.2. How Does the MetaData Printing Utility Work?

It simply attaches to the input table, opens it, reads the MetaData and prints it to the output file.

### 12.17.3. Tables Required in the Target Database

None.

### 12.17.4. Examples of Invoking the MetaData Printing Utility

The Metadata Printing Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

```
CTLU014.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SEE
TL3000
InCatalogName=SEETL3000
InSchemaName=dbo
InTableName=input_order_facts
 DebugLevel=0
WorkFileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\input_order_facts.mda
ErrorMessageOutput=cerr
Audit=No
```

Note that it opens the 'In' connections and not the 'Out' connections.

The user should keep in mind that the utility actually performs a select * from input table to retrieve the meta data so if the view it needs to open is very complicated and requires a lot of time to return the first record then this utility can require some run time to load the metadata for this view.

## *12.18. MetaData Loading Utility*

The MetaData Printing Utility proved such a hit the next question we were asked was whether we could create a utility to load the MetaData for a view into a table, rather than just print it. Of course, this was trivial and so the MetaData Loading Utility was created. This utility allows a user to load the column definitions for a view/table into a working table. The working table defaults to the name **ctl_column_defs.** However, the user can load the MetaData to any target table that meets the format of the table defined in the documentation.

### 12.18.1. Why Have the MetaData Loading Utility?

By loading the metadata for a view or both the source and the target it is possible for the customer to write their own queries or analysis on these views to check the metadata or to link the tables together in any way they please. It is just a little easier to deal with than the printed format.

### 12.18.2. What Does the MetaData Loading Utility Do?

The MetaData Printing Utility loads the metadata for a table to a target table which can then be queried like any other table.

### 12.18.3. How Does the MetaData Loading Utility Work?

It simply attaches to the input table, opens it, reads the MetaData and then loads it to the output table provided as the OutTable parameter. The table is written to by column position not by column name so the user must supply a table that had column data types as specified below.

### 12.18.4.Tables Required in the Target Database

- ctl_column_defs or similar.

The Metadata Loading Utility loads table metadata into a table so that it can be used in reports or analysis for mapping jobs. The table name can be passed as a parameter. Also, the column names are NOT used inside the program however the fields are expected to be of the following format and the first 4 columns are expected to be the primary key. It is probably advisable to use the following table definition but it is not 'required'.

Please note that the Metadata Loading Utility actually issues unconditional delete statements to remove a row from the target table before issuing the insert. This is to make it easily re-runnable and it was easier to code than to test for the violated constraint. Therefore the first 4 columns must start with "pk_" even if you choose to call them something else.

```
drop table dbo.ctl_column_defs;

create table dbo.ctl_column_defs
    (  pk_catalog_name        varchar     (255)       not null    default 'unknown'
      ,pk_schema_name         varchar     (255)       not null    default 'unknown'
      ,pk_table_name          varchar     (255)       not null    default 'unknown'
      ,pk_column_name         varchar     (255)       not null    default 'unknown'
      ,column_number          integer                 not null    default 0
      ,primary_key_ind        integer                 not null    default 0
      ,column_length          integer                 not null    default 0
      ,decimal_digits         integer                 not null    default 0
      ,nullable               integer                 not null    default 0
   )
;
```

### 12.18.5.Examples of Invoking the MetaData Loading Utility

The DataStage Job Submission Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

```
CTLU015.exe
DBConnectionInParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SEE
TL3000
InCatalogName=SEETL3000
InSchemaName=dbo
InTableName=input_order_facts
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SE
ETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=ctl_column_defs
ErrorMessageOutput=cerr
Audit=No
```

Note that it connects to the 'In' parameters for the source table and writes to the 'Out' parameters to connect to the table where the metadata will be loaded.

Of course, the best way to run this utility is as a separate command which then is invoked with the table name as a parameter. It is possible to load all metadata for an entire system like this in just a few minutes if the tables are empty.

The user should keep in mind that the utility actually performance a select * from input table to retrieve the meta data so if the view it needs to open is very complicated and requires a lot of time to return the first record then this utility can require some run time to load the metadata for this view.

## *12.19. Data Correction Utility*

One of our largest clients asked us to write a suite of data correction routines for them based on the data type of the target field when the input field was a string. Having written these routines it seems to us to be a good idea to include them into a standardised utility for other clients to use.

### 12.19.1. Why Have the Data Correction Utility?

We had originally decided that any data correction was outside the scope of the SeETL[RT] Utilities. This means our early clients have had to perform all data correction in tools outside of the SeETL[RT]. With the introduction of the Data Correction Utility many of the standard edits that must be applied to data in order to get it into one of the supported databases can be performed within the SeETL[RT] Utilities. This provides more value for our clients.

### 12.19.2. What Does the Data Correction Utility Do?

As of SeETL[RT] Utilities 3.0.00 the Data Correction Utility performs the following functions:

1. Truncate leading blanks.

2. For string fields representing decimal or numeric fields it can remove the characters ',' or '.' as directed via parameters. This is to allow the SeETL[RT] Utilities to accept string representations of numbers and to load them into numeric fields.

3. For date and timestamp fields of over 50 different formats it will reformat the data/timestamp to the ISO standard formats of YYYY-MM-DD or YYYY-MM-DD HH:MM:SS to allow it to be loaded into an SeETL[RT] supported database. Note that milliseconds are currently not supported.

4. For dates and datetime fields which are found to be invalid in the input data the user can set a default value of the date or set the date to NULL. This was a specific request from a major client. We hope others also find it useful.

### 12.19.3. How Does the Data Correction Utility Work?

On startup the Data Correction Utility reads its parameters to note what kind of data validation should be performed during this execution of the program. The Data Correction Utility then opens and reads the input file in the normal fashion. It then opens the target table describing the data types that the target data should comply to. It detects the same field names in the source and the target. As it moves data from source to target it each column it checks to see if any edits should be applied. If they should be applied it applies these edits generating data to be placed into the target table which will be valid when loaded.

The Data Correction Utility writes the output data to a file rather than to the database directly.

## 12.19.4.Data Correction Utility Parameters

Because the Data Correction Utility contains a large suite of parameters that are only available for use with this specific utility these parameters have been documented here rather than in the general parameter list.

| Parameter Name | Description and valid values |
|---|---|
| TruncateLeadingBlanks | This parameter will ensure that all fields that are copied to target fields which are character will be searched for leading blanks and all leading blanks will be removed from the string prior to being written to the output file.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'No'. |
| RemoveCommasFrom Numerics | This parameter will ensure that all fields that are copied to target fields which are numeric will be searched for the ',' character and the ',' character will be removed from the string prior to being written to the output file.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'No'. |
| RemoveDecimalsFrom Numerics | This parameter will ensure that all fields that are copied to target fields which are numeric will be searched for the '.' character and the '.' character will be removed from the string prior to being written to the output file.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'No'. |
| TranslateCommaToDotIn Numerics | This parameter will allow the user to translate the ',' character to the '.' character in a numeric field that is passed to the Data Correction Utility as a string. This is for those European countries that use '.' for the 1000s separator and ',' for the decimal place but must load the data to a database that is not supporting the same.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'No'. |
| TranslateDotToCommaIn Numerics | This parameter will allow the user to translate the '.' character to the ',' character in a numeric field that is passed to the Data Correction Utility as a string. This is for those European countries that use '.' for the 1000s separator and ',' for the decimal place but must load the data to a database that is not supporting the same.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'No'. |
| TranslateDateFormat | This parameter tells the Data Correction Utility that date format corrections will be performed for all fields in the table that are of the 'date' datatype. The user should |

| | |
|---|---|
| | be aware that databases like oracle define dates to be datetime even when the date is specified in the create table statement. Therefore, this parameter will not cause data correction to take place where the underlying database uses the datetime format to support date, as does oracle. Other databases like SQL Server and Sybase IQ, which support both dates and DateTimes may use this parameter to correct data that is being written to date fields. This parameter is used in conjunction with the 'DateFormatFrom' parameter to tell the Data Correction Utility what format the dates inside the file are expected to be in. Note. The utility currently only supports one data format for all dates inside a single input file.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'No'. |
| DateFormatFrom | This parameter tells the Data Correction Utility the date format that is expected in the specific file that is currently being processed. The Data Correction Utility will assume that all dates in the file represent valid dates and are of the format specified in this parameter and convert them to the ISO standard date format of YYYY-MM-DD so that it can be loaded by the data transfer utility into the staging area.<br><br>Note. The DateFormatFrom supports two digit years. In the case that a 2 digit year is supplied all years over '20' will be considered to be 20th century years such as a YY of 80 will be converted to 1980. Any year that is less than or equal to '20' will be considered to be a 21st century year. For example a YY of 02 will be converted to '2002'.<br><br>There are a very large number of valid values for this parameter representing all the different date formats that can be converted. These are the actual values to provide as the DateFormatFrom parameter. They are as follows:<br><br>• DD/MM/YY<br>• DD-MM-YY<br>• DD/MM/YYYY<br>• DD-MM-YYYY<br>• DD/MMM/YY<br>• DD-MMM-YY<br>• DD/MMM/YYYY<br>• DD-MMM-YYYY<br>• MM/DD/YY<br>• MM-DD-YY<br>• MM/DD/YYYY<br>• MM-DD-YYYY<br>• MMM/DD/YY<br>• MMM-DD-YY<br>• MMM/DD/YYYY<br>• MMM-DD-YYYY<br>• YY/MM/DD<br>• YY-MM-DD<br>• YYYY/MM/DD<br>• YYYY-MM-DD<br>• YY/MMM/DD<br>• YY-MMM-DD<br>• YYYY/MMM/DD<br>• YYYY-MMM-DD<br><br>Default Value:<br>"Not Set" |
| TranslateDateTimeFormat | This parameter tells the Data Correction Utility that datetime (timestamp) format |

| | |
|---|---|
| | corrections will be performed for all fields in the table that are of the timestamp datatype. This parameter is used in conjunction with the 'DateTimeFormatFrom' parameter to tell the Data Correction Utility what format the timestamps inside the file are expected to be in. Note. The utility currently only supports one timestamp format for all timestamps inside a single input file.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'No'. |
| DateTimeFormatFrom | This parameter tells the Data Correction Utility the timestamp format that is expected in the specific file that is currently being processed. The Data Correction Utility will assume that all timestamps in the file represent valid timestamps and are of the format specified in this parameter and convert them to the ISO standard timestamp format of YYYY-MM-DD HH:MM:SS so that it can be loaded by the data transfer utility into the staging area.<br>There are a very large number of valid values for this parameter representing all the different date formats that can be converted.<br><br>Note 1. The DateTimeFormatFrom supports two digit years. In the case that a 2 digit year is supplied all years over '20' will be considered to be 20th century years such as a YY of 80 will be converted to 1980. Any year that is less than or equal to '20' will be considered to be a 21st century year. For example a YY of 02 will be converted to '2002'.<br><br>Note 2. The DateTimeFormatFrom supports imbedded blanks inside the standard formats for datetimes. In order to pass these formats as a parameter to the Data Correction Utility blanks are represented by the lower case letter 'b'.<br><br>They are as follows:<br><br>• DD/MM/YYbHH:MM:SS<br>• DD/MM/YYYYbHH:MM:SS<br>• DD.MM.YYbHH:MM<br>• DD/MMM/YYbHH:MM:SS<br>• DD/MMM/YYYYbHH:MM:SS<br>• DD/MMM/YYYY:HH:MM:SS<br>• DD-MM-YYbHH:MM:SS<br>• DD-MM-YYYYbHH:MM:SS<br>• DD-MMM-YYbHH:MM:SS<br>• DD-MMM-YYYYbHH:MM:SS<br>• MM/DD/YYbHH:MM:SS<br>• MM/DD/YYYYbHH:MM:SS<br>• MMM/DD/YYbHH:MM:SS<br>• MMM/DD/YYYYbHH:MM:SS<br>• MM-DD-YYbHH:MM:SS<br>• MM-DD-YYYYbHH:MM:SS<br>• MMM-DD-YYbHH:MM:SS<br>• MMM-DD-YYYYbHH:MM:SS<br>• YY/MM/DDbHH:MM:SS<br>• YYYY/MM/DDbHH:MM:SS<br>• YY/MMM/DDbHH:MM:SS<br>• YYYY/MMM/DDbHH:MM:SS<br>• YY-MM-DDbHH:MM:SS<br>• YYYY-MM-DDbHH:MM:SS<br>• YY-MMM-DDbHH:MM:SS<br>• YYYYMMDDHHMMSSZ<br>• YYYY-MMM-DDbHH:MM:SS<br>• YYYY-MM-DD-HH.MM.SS.MMMMMM |

| | |
|---|---|
| | Default Value:<br>“Not Set” |
| DefaultInvalidDates | SeETL was originally written based on the definition that all data coming into SeETL would be valid. That is, the data that the user requested to be placed into a field could be placed into that field using the ODBC implicit data conversion available in the specific ODBC driver being employed.<br><br>SeETL was never intended to be able to perform data correction. Our recommendation in this area was where it was possible for data to be invalid place it into a string and then validate it though a stored procedure of other processing capability.<br><br>The main reason for taking this approach is that the validation rules for all the data types and the defaulting rules for all those values is a large amount of code to write and difficult to put into parameters.<br><br>That said, we have had a number of clients ask us to be able to at least default invalid dates. The reason being that a lot of operational system set ‘null’ or unknown date to ‘0’ or blank. Obviously, such validation could be performed in stored procedures or in the views that extract dates from the staging area. However, the problem of invalid dates is so common that we have decided to put date default processing into the Data Correction Utility.<br><br>The way this works is that the user may specify the correction of both Dates and DateTimes. When such checking is requested every Date/DateTime field is checked to see if it is valid to be placed into the database that the user is using. Should the Date/DateTime be invalid the Date/DateTime will be set to the value in the Default Date/DateTime parameter. Note that no validation is performed on the Default Date/DateTime parameter.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>‘No’. |
| DefaultInvalidDateTimes | This parameter tells the Data Correction Utility to default any invalid DateTimes to the Default DateTime parameter.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>‘No’. |
| DefaultDate | This parameter is the default date to be used in the case where a date is found to be invalid. It must be a date in the string format that is valid to be converted to a date by the implicit data conversion capability of your ODBC driver. The usual format of such dates is ‘YYYY-MM-DD’.<br><br>Please note. The capability of the implicate data conversion for a character string to a date varies by ODBC driver and you should consult yoyur ODBC driver documentation for further details.<br><br>Note. Some clients want to default invalid dates to NULL. This has been allowed as a valid parameter.<br><br>Valid values are:<br>• ‘YYYY-MM-DD’ valid date string for your database.<br>• NULL |

| | |
|---|---|
| | Default Value:<br>'NULL'. |
| DefaultDateTime | This parameter is the default DateTime to be used in the case where a DateTime is found to be invalid. It must be a DateTime in the string format that is valid to be converted to a DateTime by the implicit data conversion capability of your ODBC driver. The usual format of such DateTime is 'YYYY-MM-DD HH:MM:SS.mmm' where mmm is milliseconds. Note that milliseconds are only supported by some databases and you should consulting your database documentation on it's capabilities for storing DateTimes to determine if you must supply default values for milliseconds.<br><br>Please note. The capability of the implicate data conversion for a character string to a date varies by ODBC driver and you should consult yoyur ODBC driver documentation for further details.<br><br>Note. Some clients want to default invalid dates to NULL. This has been allowed as a valid parameter.<br><br>Valid values are:<br>• 'YYYY-MM-DD' valid date string.<br>• NULL<br><br>Default Value:<br>'NULL'. |

### 12.19.5.Tables Required in the Target Database

- None.

### 12.19.6.Examples of Invoking the Data Correction Utility

The Data Correction Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

```
CTLU016.exe
DBConnectionOutParameter=DSN=SEETL3000;SERVER=PETERLAP;UID=dba;PWD=password;DATABASE=SE
ETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=f_order_fact
CTLF001FileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF001.DAT
CTLF002FileName=D:\IBISoftware\SeETL\DesignTime\3.0.00\Data\CTLF002.DAT
TruncateLeadingBlanks=Yes
RemoveCommasFromNumerics=Yes
RemoveDecimalsFromNumerics=Yes
TranslateDateFormat=Yes
TranslateDateTimeFormat=Yes
DateFormatFrom=DD-MM-YY
DateTimeFormatFrom=DD/MMM/YYbHH:MM:SS
ErrorMessageOutput=cerr
Audit=No
DefaultInvalidDates=Yes
DefaultInvalidDateTimes=Yes
DefaultDate=1900-01-01
DefaultDateTime="1900-01-01 00:00:00"
```

Note that this utility only connects to the 'Out' parameters to determine the data types of the target fields for the OutTableName. This table is not actually written to during processing. The output from the utility is written to CTLF002FileName.

## *12.20. Batch Processing Sleep Utility*

On windows there is no standard utility that allows you to sleep a batch for a specific period of time. There are utilities that can be downloaded and integrated into the schedule. However, a client asked us for a program that would allow them to put a sleep into the batch using standard utilities.

### 12.20.1. Why Have the Batch Processing Sleep Utility?

There is no single standard way of sleeping a batch over all the operating systems that SeETL runs on. A client asked for a single mechanism to implement sleeps in the batch schedule so we created this utility.

### 12.20.2. What Does the Batch Processing Sleep Utility Do?

The Batch Processing Sleep Utility simply goes to sleep for BatchSleepSeconds. It will then complete.

### 12.20.3. How Does the Batch Processing Sleep Utility Work?

It is written to be sensitive to the operating system on which it runs. It invokes the appropriate sleep utility from the operating system on which it is running.

### 12.20.4. Tables Required in the Target Database

None.

### 12.20.5. Examples of Invoking the Batch Processing Sleep Utility

The Batch Processing Sleep Utility is called using the same command line interface as all other SeETL$^{RT}$ Utilities. You could invoke it as follows:

```
CTLU017.exe
DBConnectionOutParameter=DSN=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=not_used
ErrorMessageOutput=cerr
DebugLevel=0
Audit=Yes
BatchSleepSeconds=15
```

The program will start and then sleep for BatchSleepSeconds.

## *12.21. XML File Reformat Utility*

The amount of XML based data is increasing. And companies are starting to need to load XML data into their data warehouses. There are a lot of XML covenrters out there and most of our clients have been satisfied with using an XML converter and then reformatting the data to be loaded. This has required the use of a third party XML converter. We were asked by a client if we could write a relatively simple XML converter to convert XML data to the SeETL internal file format.

### 12.21.1.Why Have the XML File Reformat Utility?

The XML File Reformat Utility allows simple translation of simple XML documents to occur inside the SeETL tool suite with the normal logging of the processing. It is simply a utility that was requested and written for a specific case to make data easily loadable into a clients data warehouse.

The tool is quite simple. Should you require more complex XML transformations please contact support@instantbi.com to outline your requirements.

### 12.21.2.What Does the XML File Reformat Utility Do?

The XML File Reformat Utility takes a simple input XML file and produces an transformed file using the SeETL Internal File Format format.

The following example provides an indication of what the utility can do. The following data is a small snippet of a simple file where there is a definition for the overall file and a set of elements with attributes on the elements. It is simply linking product ids to article ids. Nothing too complicated.

```
<ns0:PurchaseArticleData xmlns:ns0="http://namespacedata" DataType="PURCHASEDATA" SeqNr="1"
Date="11.09.2007" ProcessId="132950">
   <PurchaseSku purchaseArticleProductId="PAR_100425" articleKey="22575" fromDate="12.07.2006"/>
   <PurchaseSku purchaseArticleProductId="PAR_101221" articleKey="22740" fromDate="12.07.2006"/>
   <PurchaseSku purchaseArticleProductId="PAR_101280" articleKey="217670" fromDate="12.07.2006"/>
</ns0:PurchaseArticleData>
```

This data is wanted to be loaded into a table. That table is defined as follows:

```
create table dbo.PurchaseArticleData
   (  namespace_field          varchar   (255)        not null default 'unknown'
   ,  DataType                 varchar   (255)        not null default 'unknown'
   ,  SequenceNumber           varchar   (255)        not null default 'unknown'
   ,  ProcessingDate           varchar   (255)        not null default 'unknown'
   ,  ProcessingNumber         varchar   (255)        not null default 'unknown'
   ,  ProductId                varchar   (255)        not null default 'unknown'
   ,  ArticleKey               varchar   (255)        not null default 'unknown'
   ,  FromDate                 varchar   (255)        not null default 'unknown'
   )
;
```

The target table is used to detect the column names and data types of the target columns the same way as it is always used on other utilities.

The data that is produced is then as follows:

8~namespace_field~0~12~255~0~0~DataType~0~12~255~0~0~SequenceNumber~0~12~255~0~0~ProcessingDate~0~12~255~0~0~ProcessingNumber~0~12~255~0~0~ProductId~0~12~255~0~0~ArticleKey~0~12~255~0~0~FromDate~0~12~255~0~0~
8~http://namespacedata~0~PURCHASEDATA~0~1~0~11.09.2007~0~132950~0~PAR_100425~0~22575~0~12.07.2006~0~
8~http://namespacedata~0~PURCHASEDATA~0~1~0~11.09.2007~0~132950~0~PAR_101221~0~22740~0~12.07.2006~0~
8~http://namespacedata~0~PURCHASEDATA~0~1~0~11.09.2007~0~132950~0~PAR_101280~0~217670~0~12.07.2006~0~

You can see that the columns have been moved from the source xml file to the target SeETL internal file format file.

The source code for the program is published if you would like to see the details of what it does and how it does it's work. Basically it reads the 0 level element that is used to define the file and the attributes of that element. It then reads the individual element and the data in the attributes of the individual elements.

As output it strings together the attributes from the 0 level element and the attributes from each element line.

The data can be moved to the output file as either by name or by column position movements.

### 12.21.3.How Does the XML File Reformat Utility Work?

It simply opens the input file, loads it into an xml document, opens the target table, and then moves the data across for each row in the XML file.

### 12.21.4.Tables Required in the Target Database

- OutTable.

The XML File Reformat Utility reformats data from an XML file by column name or by column position. In order to do this the OutTable must exist in the database pointed to by the TargetODBCConnection.

### 12.21.5.Examples of Invoking the XML File Reformat Utility

The XML File Reformat Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

CTLU018.exe
DBConnectionOutParameter=DSN=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=PurchaseArticleData
CTLF001FileName=C:\IBISoftware\SeETL\DesignTime\3.0.02\TestData\test1.xml
CTLF002FileName=C:\IBISoftware\SeETL\DesignTime\3.0.02\TestData\test1.xml.dat
ErrorMessageOutput=cerr
DebugLevel=9
MoveByColumnName=No
MoveByColumnPosition=Yes

## *12.22. Delete Target Rows Utility*

The Data Transfer Utility has an option to be able to delete rows that already exist prior to insertion. We also introduced the ability to delete rows while building a Load Interface File. We then introduced the ability to respect sequence numbers for rows for fact tables. With the introduction of the ability to respect sequence numbers we introduced the issue that if the Data Transfer Utility was run and was selecting sequence numbers of rows and then those rows were being deleted and the program failed there would be no way to get those sequence numbers back.

In short, the Data Transfer Utility was not re-runnable from the beginning when creating a Load Interface File and deleting the rows that previously existed.

Hence the Delete Target Rows Utility was needed to make the entire process re-runnable. The Data Transfer Utility can write a file for rows to be deleted. This utility can then be run to delete the rows. Therefore each step is restartable from the beginning.

### 12.22.1. Why Have the Delete Target Rows Utility?

The Delete Target Rows Utility is needed to make the ability to respect sequence numbers on fct tables while producing a Load Interface File re-runnable from the beginning of each step.

### 12.22.2. What Does the Delete Target Rows Utility Do?

The Delete Target Rows Utility reads a file named CTLF0001 and uses the records in this file to issue deletes against the OutTable.

### 12.22.3. How Does the Delete Target Rows Utility Work?

The Delete Target Rows Utility simply reads CTLF0001 and for each row in this file it issues a delete against the target OutTable. It is a very simple utility.

### 12.22.4.Tables Required in the Target Database

The following table must exist in the target database.

- OutTable.

The Delete Target Rows Utility deletes rows from the OutTable therefore the OutTable must exist in the target database.

### 12.22.5.Examples of Invoking the Delete Target Rows Utility

The Delete Target Rows Utility Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

```
CTLU019.exe
DBConnectionOutParameter=DSN=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=f_order_fact
CTLF001FileName=C:\IBISoftware\SeETL\DesignTime\3.0.02\TestData\f_order_fact.dat
ErrorMessageOutput=cerr
DebugLevel=9
AutoCommit=No
CommitFrequency=10000
```

## *12.23. Bulk Load Rows into SQL Server Utility*

The suite of SeETL utilities has been designed to be database independent. The specific databases implement things like the loaders slightly differently and so we did not write to the bulk load interface for the various databases. This works fine for all databases except SQL Server when using partitioned clustered tables.

When using SQL Server with partitioned clustered tables the performance of the deleting and loading is relatively poor compared to other databases. So, as we were working on larger SQL Server databases we decided to write a specific utility to take advantage of the bulk loader for SQL Server as well as taking advantage of performing bulk operations on the clustered tables so that the work could be pushed into the database engine rather than being executed on row at a time as per normal.

Should you wish to have a similar utility for other databases please write to support@instantbi.com with your request.

### 12.23.1.Why Have the Bulk Load Rows into SQL Server Utility?

The Bulk Load Rows into SQL Server Utility is used simply used to speed up the processing of putting rows into partitioned clustered fact tables for SQL Server 2005.

It also allows for fewer steps in the schedule. This is at the cost of the restartability to be slightly less granular. The program can only be restarted at the beginning and therefore some rework might be performed when restarting the program.

### 12.23.2.What Does the Bulk Load Rows into SQL Server Utility Do?

The Bulk Load Rows into SQL Server Utility peforms the following functions:

1. It truncates the output temporary table to make sure there are no rows in it.
2. It performs a bulk load of the Load Interface File that is passed to the program into the temporary table.
3. It then uses the keys passed as parameters to the program to perform a bulk delete of any rows that exist in the target table that will be replaced by the rows in the temporary table.
4. Once the rows that will be replaced are deleted it performs a insert into select * from the temporary table.

By performing the delete and the insert as single statements the optimizer has a better chance to optimize the processing and to perform bulk sorts etc in the partitioning of the data going into the target table as well as sorting for the insert which can be based on the clustering sequence of the target table.

### 12.23.3.How Does the Bulk Load Rows into SQL Server Utility Work?

The Bulk Load Rows into SQL Server Utility works as follows:

1. It is passed the name of the Load Interface File, the name of the target table, and the names of the key fields that need to be used for the creation of the delete and insert statements.
2. A number of sql statements are built to be performed against the database directly by connecting through the ODBC connection.
3. The 4 statements that were constructed are then executed against the database.

### 12.23.4.Tables Required in the Target Database

The following table must exist in the target database.

- OutTable.
- OutTable_tmp.
- OutTable_ins.

The Bulk Load Rows into SQL Server Utility loads rows into the OutTable_tmp table using the bulk loader and then uses delete and insert operations to move the data from OutTable_tmp to OutTable_ins. The OutTable_ins table is usually based on the OutTable as a matter of naming standards.

### 12.23.5.Examples of Invoking the Bulk Load Rows into SQL Server Utility

The Bulk Load Rows into SQL Server Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

```
CTLU020.exe
DBConnectionOutParameter=DSN=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=f_order_fact
CTLF001FileName=C:\IBISoftware\SeETL\DesignTime\3.0.02\TestData\CTLF007.LIF
ErrorMessageOutput=cerr
DebugLevel=0
BulkInsertParameters="DATAFILETYPE='char',FIELDTERMINATOR=',',FIRSTROW=2,CHECK_CONSTRAINTS,K
EEPNULLS"
UniqueKeyFieldName1=pk_time_dim
UniqueKeyFieldName2=pk_product_dim
UniqueKeyFieldName3=pk_customer_dim
UniqueKeyFieldName4=pk_vendor_dim
```

### 12.23.6.Bulk Load Rows into SQL Server Utility Parameters

Because the Bulk Load Rows into SQL Server Utility contains a number of parameters specific to this program we have documented the parameters separately.

| Parameter Name | Description and valid values |
|---|---|
| BulkInsertParameters | This parameter is the set of parameters that will be passed to the SQL Server Bulk Insert command. It is a mandatory parameter and an error message will be generated if it is not provided.<br><br>Valid values are:<br>• Any valid set of parameters for the SQL Server bulk loader.<br><br>Default Value:<br>'Not Set'. |
| UniqueKeyFieldName1- 20 | When generating the bulk commands to perform the deletes and inserts a set of key fields must be provided to generate the sql commands. We have allowed for up to 20 fields to be specificied as the key fields for the delete and insert statement.<br><br>Valid values are: |

| | |
|---|---|
| | • Names of the UniqueKeyFileNames for the OutTable.<br><br>Default Value:<br>'Not Set'. |

## *12.24. Execute ProcessName From Scheduler Utility*

When testing SeETL jobs we have always created command files and executed those command files. Once the processing was tested we then embedded the commands into the scheduler and ran the commands from inside the scheduler from the ctl_process_commands table.

This all works fine with the exception that when one makes changes to the commands inside the SeETL workbook one has to also remember to make the same changes to the test commands. If one forgets then occasionally the testing results do not property match the results produced by running the scheduler.

One of our clients asked us if we could develop a command where they could issue a request to run a command from the ctl_process_commands table in the target database from the command line.

The way this was developed was to enable the passing of parameters into the command as well as the ability to over-ride the parameters that are not adjustable inside the command by merely appending them to the command that is entered at the command line interface.

### 12.24.1.Why Have the Execute ProcessName from Scheduler Utility?

The main reason to have the Execute ProcessName From Scheduler Utility is to remove the need for two copies of the commands to be executed, one in the database for the scheduler and one for the testing of commands at the command line interface.

### 12.24.2.What Does the Execute ProcessName from Scheduler Utility Do?

The Execute ProcessName From Scheduler Utility

1. Accepts a number of parameters including the command that is to be run.
2. It then retrieves the ProcessName command from the ctl_process_commands table in the target database.
3. It then perform parameter substitution for the parameters passed to the program to be put into the command string to be submitted to the command processor.
4. It then executes the command via the command processor.

### 12.24.3.How Does the Execute ProcessName from Scheduler Utility Work?

Execute ProcessName From Scheduler Utility simply decodes the parameters that are passed and performs parameter substitution before passing the parameter to the command processor.

### 12.24.4.Tables Required in the Target Database

- ctl_process_commands

The Execute ProcessName From Scheduler Utility uses the ctl_process_commands table to retrieve the command to be executed.

## 12.24.5.Examples of Invoking the Execute ProcessName from Scheduler Utility

The Execute ProcessName From Scheduler Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

```
CTLU021.exe
ProcessName=testcommand
ProcessNameParameter01=ProcessNameParameter01
ProcessNameParameter02=ProcessNameParameter02
ProcessNameParameter03=ProcessNameParameter03
DBConnectionOutParameter=DSN=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=f_order_fact
CTLF001FileName=C:\IBISoftware\SeETL\DesignTime\3.0.02\TestData\CTLF007.LIF
 ErrorMessageOutput=cerr
DebugLevel=0
AppendOverRideParameterList=Yes
```

Note that when you pass parameters which are not substituted they will be appended to the command string that will be submitted to the command processor and therefore they will over-ride the same parameters that may already be specified in the command in the ctl_process_commands table.

This is because the parameter processing for SeETL will take the last specification of a parameter in a command submitted to the command line.

## 12.24.6.Execute ProcessName from Scheduler Utility Parameters

Because the Execute ProcessName From Scheduler Utility contains a number of parameters specific to this program we have documented the parameters separately.

| Parameter Name | Description and valid values |
|---|---|
| ProcessName | This parameter is the name of the ProcessName to be executed from the ctl_process_commands table in the target database. It is a mandatory parameter and an error message will be generated if it is not provided.<br><br>Valid values are:<br>• Any ProcessName<br><br>Default Value:<br>'Not Set'. |
| ProcessNameParameter01 - 10 | The ProcessName may have a number of parameter that are able to be supplied as part of the ProcessName command. A ProcessName can have up to 10 parameters.<br><br>Valid values are:<br>• Any valid parameter<br><br>Default Value:<br>'Not Set'. |
| AppendOverRide ParameterList | This parameter takes some explaining. When running a command from the ctl_process_commands table the user may want to over-ride some of the parameters to the command which are not parameterised in the command itself b positional parameters denominated by the ?n notation. For example in all production commands DebugLevel=0 will be set and the NumberRowsToTransfer will be absent. They are not appropriate in production. |

| | |
|---|---|
| | However, in testing it is normal for these parameters to be present and set. So a mechanism is required to be able to set them. This is done by merely passing the parameters at the end of the parameter string to CTLU021 and setting AppendOverRideParameterList=Yes. This will pass these parameters to the program that is called in the command in ctl_process_command.<br><br>Note. In order for the Execute ProcessName From Scheduler Utility to know where to read the ctl_process_command table from the information about which database to connect to must be passed to the Execute ProcessName From Scheduler Utility. So these parameters must always be present.<br><br>There are some commands where these parameters are not appropriate to be passed to the command. For example commands to copy or delete files must not have the extra parameters required to run the Execute ProcessName From Scheduler Utility passed to them. In these cases the user must set AppendOverRideParameterList=No.<br><br>Valid values are:<br>• Yes<br>• No<br><br>Default Value:<br>'Yes'. |

## *12.25. Batch Processing Scheduling Utility – Resilient Version*

### 12.25.1.Why Have the Batch Processing Scheduling Utility – Resilient Version?

The first version of the SeETL[RT] batch processing scheduling utility was intended to run job on the same machine as the data warehouse resided. It was thought that occasionally the staging area might be on another machine, in which case it was theorised that another scheduler would run on that machine. None of the SeETL[RT] programs are resilient to network failures. If the connection to the database is lost the program fails. This is acceptable with all OTHER programs as the schedule can simply be restarted. However, for the scheduler, when the scheduler fails, it then needs to be restarted. One client runs SeETL[RT] as a hub on a large ETL Server. There are, inevitably, occasional flickers of the network and since the database that the scheduler is looking at resides on a different machine these 'flickers' cause loss of network connection and failure of the scheduler.

Efforts were made to upgrade the existing scheduler but they proved fruitless. In the end it was decided to copy and make substantial changes to the scheduler to gain a higher level of resilience.

This was achieved by altering the program to make a connection every time it checks the schedule. The program now closes all ODBC connections before going to sleep for BatchSleepSeconds. When it wakes up again it re-establishes all connections and then searches the schedule to see what state the schedule is in now. It assumes NO memory of any of the data that it knew about previously and so must re-read the schedule and the logs to see what has happened since it was last 'awake'.

It was determined not to upgrade the current scheduler to perform such processing. After all, when the scheduler runs on the machine where the schedule resides all this is not necessary. There are very rarely any problems with the 'network connection' to the local database. Some clients have run the scheduler for months with no outage on linux and unix machines. Because this problem is so rare and affects so few accounts we decided to write a separate utility for that one client and make it available for other clients should they wish to avail of it.

Writing such schedulers is not an easy thing to do. And the scheduler is easily the most important and most sensitive to change program of all of those we release. A 'problem' with the scheduler affects everyone. Reliability comes in front of 'rarely used new features'.

## *12.26. Execute SQL Statement and Logging Utility*

With the further development of SeETL[DT] to be able to create SQL fragments that can be executed against the database we have developed a more robust and functional SQL execution utility. The prior version, which will remain available for existing customers, was merely intended to be able to execute small snippets of SQL that were a minor portion of the ETL subsystem.

Now that we are building entire ETL subsystems using SQL an SQL execution facility that included logging and query ability on start, stop times, the file it was loaded from etc was necessary.

### 12.26.1. Why Have the Execute SQL Statement and Logging Utility?

The main reason to have the Execute SQL Statement and Logging Utility is to provide a greater level of security, auditability, tracability to the newly developed ability to generate SQL for the ETL subsystem.

The use of the file system was simply not acceptable as the way of storing the SQL that is generated. It needs to be in a repository of the client like Tortoise SVN or it needs to be in a database.

File systems are notoriously bad for storing source code when it has no contols placed around it.

### 12.26.2. What Does the Execute SQL Statement and Logging Utility Do?

The Execute SQL Statement and Logging Utility

1. Accepts an input parameter called SQLFileName which indicated the file name from which the SQL to be executed was loaded.
2. It then retrieves the SQL statement(s) from the ctl_dw_sql_stmts table. If the statement has been broken up to fit into the table it is pasted back together.
3. It then retrieves the start timestamp, submits the SQL statement to the database, and retrieves the stop timestamp.
4. It then writes the details of execution of the statement to the ctl_dw_sql_stmt_log.

### 12.26.3. Tables Required in the Target Database

- ctl_dw_sql_stmts
- ctl_dw_sql_stmt_log

The Execute SQL Statement and Logging Utility uses the ctl_dw_sql_stmts table to retrieve the SQL statements to be executed. It uses the ctl_dw_sql_stmt_log to write the log of the statements.

This is the sql statements table.

```
create table dbo.ctl_dw_sql_stmts (
        pk_spreadsheet_name              varchar (255)       default 'unknown'
       ,pk_spreadsheet_version           varchar (255)       default 'unknown'
       ,pk_row_number                    integer             not null default 0
       ,pk_load_timestamp                datetime            not null default current_timestamp
       ,current_flag                     integer             not null default 0
       ,directory_name                   varchar (4000)      not null default 'unknown'
       ,sql_file_name                    varchar (4000)      not null default 'unknown'
       ,full_sql_file_name               varchar (4000)      not null default 'unknown'
       ,sql_file_win_user                varchar (255)       default 'unknown'
       ,sql_file_win_workstation         varchar (255)       default 'unknown'
       ,sql_file_win_domain              varchar (255)       default 'unknown'
       ,sql_file_os_version              varchar (255)       default 'unknown'
       ,sql_file_seetl_current_directory varchar (4000)      default 'unknown'
       ,sql_file_dot_net_version         varchar (255)       default 'unknown'
       ,sql_file_seetl_version_date      varchar (255)       default 'unknown'
       ,sql_statement_01                 varchar (8000)      not null default 'unknown'
       ,sql_statement_02                 varchar (8000)      not null default 'unknown'
       ,sql_statement_03                 varchar (8000)      not null default 'unknown'
       ,sql_statement_04                 varchar (8000)      not null default 'unknown'
)
```

The following table explains the values of each column and the functions available.

| Column Name | Function |
|---|---|
| pk_spreadsheet_name | This is the name of the spreadsheet that is used to cause the sql statement to be loaded into this table. |
| pk_spreadsheet_version | This is the version of the spreadsheet that is used to cause the sql statement to be loaded into this table. |
| pk_row_number | Each load action is given a sequential row number starting from zero. |
| pk_load_timestamp | This is the timestamp of the row being loaded into the database. The timestamp makes the key unique. |
| current_flag | Rows may never be deleted from this table by the SeETL application. Only an administrator to the table may perform deletes if that is how the security of the table is set. |
| directory_name | This is the name of the directory that the sql in the file is loaded from. The directory is stored separately so that it is possible to easily build reports based on directory. |
| sql_file_name | This is the name of the file that the sql is loaded from. |
| full_sql_file_name | The full directory and file name is also stored to make it simple for the program to perform a direct match on this field with the SQLFileName parameter. |
| sql_file_win_user | This is the name of the authenticated windows user that was running the SeETL application to load this sql from the file into the database. |
| sql_file_win_workstation | This is the name of the workstation that was running the SeETL application to load this sql from the file into the database. |
| sql_file_win_domain | This is the name of the authenticated windows domain that was running the SeETL application to load this sql from the file into the database. |
| sql_file_os_version | This is the name of the authenticated windows operating system that was running the SeETL application to load this sql from the file into the database. |
| sql_file_seetl_current_directory | This is the name of the directory out of which SeETL was running when it loaded the statement into the database. |
| sql_file_dot_net_version | This is the name of the authenticated windows dot net version that was running the SeETL application to load this sql from the file into the database. |
| sql_file_seetl_version_date | This is the build date of the SeETL version so that the client can tell exactly what version of SeETL they are running. We have not typically reversioned SeETL every time we have made updates. |
| sql_statement_01 | This is the first 8000 characters of the SQL Statement. |
| sql_statement_02 | This is the second 8000 characters of the SQL Statement. |
| sql_statement_03 | This is the third 8000 characters of the SQL Statement. |
| sql_statement_04 | This is the fourth 8000 characters of the SQL Statement. |

```
create table dbo.ctl_dw_sql_stmt_log (
       pk_spreadsheet_name       varchar (255)       default 'unknown'
      ,pk_spreadsheet_version    varchar (255)       default 'unknown'
      ,pk_row_number             integer             not null default 0
      ,pk_load_timestamp         datetime            not null  default current_timestamp
      ,pk_batch_number           integer             not null
      ,pk_start_stmt_timestamp   datetime            not null   default current_timestamp
      ,end_stmt_timestamp        datetime            not null   default current_timestamp
      ,directory_name            varchar (4000)      not null default 'unknown'
      ,sql_file_name             varchar (4000)      not null default 'unknown'
      ,full_sql_file_name        varchar (4000)      not null default 'unknown'
      ,number_rows_affected      integer             not null default 0
      ,sql_code                  integer             not null default 0
      ,sql_statement             varchar (8000)      not null default 'unknown'
)
```

The following table explains the values of each column and the functions available.

| Column Name | Function |
|---|---|
| pk_spreadsheet_name | This is the name of the spreadsheet that is used to cause the sql statement to be loaded into this table. |
| pk_spreadsheet_version | This is the version of the spreadsheet that is used to cause the sql statement to be loaded into this table. |
| pk_row_number | Each load action is given a sequential row number starting from zero. |
| pk_load_timestamp | This is the timestamp of the row being loaded into the database. The timestamp makes the key unique. |
| pk_batch_number | This is the number of the batch that the SQL statement was executed in. By storing this it will be possible to observe the differences in elapsed times or rows processed across different batches. |
| pk_start_stmt_timestamp | The start timestamp of the sql statement. It is made part of the key to make the key unique. |
| end_stmt_timestamp | The stop timestamp of the statement. Therefore the elapsed time of the statement can be worked out for reporting purposes. |
| directory_name | This is the name of the directory that the sql in the file is loaded from. The directory is stored separately so that it is possible to easily build reports based on directory. |
| sql_file_name | This is the name of the file that the sql is loaded from. |
| full_sql_file_name | The full directory and file name is also stored to make it simple for the program to perform a direct match on this field with the SQLFileName parameter. |
| number_rows_affected | This is the number of rows affected by the SQL statement as returned by the database manager. |
| sql_code | This is the SQL Code returned by the database manager. |
| sql_statement | This is the first 8,000 bytes of the SQL statement purely for logging purposes. |

### 12.26.4.Examples of Invoking the Execute SQL Statement and Logging Utility

The Execute ProcessName From Scheduler Utility is called using the same command line interface as all other SeETL[RT] Utilities. You could invoke it as follows:

```
CTLU023.exe
DBConnectionOutParameter=DSN=SEETL3000
OutCatalogName=SEETL3000
OutSchemaName=dbo
OutTableName=not_applicable
SQLFileName="C:\temp\dmltype1\dim1p_z01_vm_address_01_dim1s_01.sql"
ErrorMessageOutput=cerr
DebugLevel=0
```

Note that the parameter SQLFileName is passed into the program and it represents the name of the file that was loaded into the database. The SQL itself is NOT read from this file by this program. It is expected to be loaded into the database by one of the administrators of the ETL subsystem.