Portrait
Foundation

DataMart Creation Utility User Guide

Edition 19.0

22 November 2013

Pitney Bowes
Software

Portrait Foundation
DataMart Creation Utility User Guide

Acknowledgement of trademarks

About Portrait Software

Portrait Software is now part of Pitney Bowes Software Inc.

Portrait Software enables organizations to engage with each of their customers as individuals, resulting in improved customer profitability, increased retention, reduced risk, and outstanding customer experiences. This is achieved through a suite of innovative, insight-driven applications which empower organizations to create enduring one-to-one relationships with their customers.

Portrait Software was acquired in July 2010 by Pitney Bowes to build on the broad range of capabilities at Pitney Bowes Software for helping organizations acquire, serve and grow their customer relationships more effectively. The Portrait Customer Interaction Suite combines world leading customer analytics, powerful inbound and outbound campaign management, and best-in-class business process integration to deliver real-time customer interactions that communicate precisely the right message through the right channel, at the right time.

Our 300 + customers include industry-leading organizations in customer-intensive sectors. They include 3, AAA, Bank of Tokyo Mitsubishi, Dell, Fiserv Bank Solutions, Lloyds Banking Group, Merrill Lynch, Nationwide Building Society, RACQ, RAC WA, Telenor, Tesco Bank, T-Mobile, Tryg and US Bank.

Pitney Bowes Software Inc. is a division of Pitney Bowes Inc. (NYSE: PBI).

For more information please visit: http://www.pitneybowes.co.uk/software/

UK	**America**	**Norway**
Portrait Software	Portrait Software	Portrait Software
The Smith Centre	125 Summer Street	Portrait Million Handshakes AS
The Fairmile	16th Floor	Maridalsveien. 87
Henley-on-Thames	Boston, MA 02110	0461 Oslo
Oxfordshire, RG9 6AB, UK	USA	Norway
Email: support@portraitsoftware.com	Email: support@portraitsoftware.com	Email: support@portraitsoftware.com
Tel: +44 (0)1491 416778	Tel: +1 617 457 5200	Tel: +47 22 38 91 00
Fax: +44 (0)1491 416601	Fax: +1 617 457 5299	Fax: +47 23 40 94 99

About this document

Purpose of document

This document describes the Portrait Foundation DataMart, its creation utility and the extension framework.

Intended audience

Any person wishing to understand the Portrait Foundation DataMart.

Related documents

None

Software release

Portrait Foundation 5.0 or later.

Contents

1 Introduction

Portrait Foundation's operational database is a complex meta-data driven data structure that is not easy to understand or interrogate. As a result, the operational database is unlikely to be suitable for data activities such as business intelligence and data extract.

Consequently Portrait Foundation offers an extract mechanism to create an alternative data repository commonly referred to as the DataMart. The DataMart will be an implementation specific logical representation of the Portrait Foundation data and thus provides the user-friendly and intuitive view that many data related activities require.

An extract utility is provided to create the DataMart and an incremental update process enables implementations to top-up the repository with the latest data changes.

This document introduces details a number of features, components and issues that are core to the DataMart and its creation process including:-

- An overview of the DataMart data structures
- An overview of the data architecture and the process for creating and incrementally updating the DataMart
- A guide on how to use the creation utility
- An implementation extension guide
- Troubleshooting tips

2 Data structure

This section discusses how the operational data model is represented in the DataMart and thus shows how the logical business entities are depicted.

Essentially there are two kinds of tables in the operational database and thus there are two kinds of tables in the DataMart.

1. Configurable tables – these are the tables that users can configure via the configuration suite. These include parties, contracts and repeating attributes.
2. Fixed tables – these are not configurable, they are system defined. The engagement table is a prominent example.

2.1 Configurable tables

The configurable tables are transformed from ambiguously named meta-data driven tables into tables that represent the logical entities. This section uses parties as an example.

A typical party tree may be:-

BaseParty

- UserName
- Password

Individual

- First Name
- Surname

Agent	Consumer
- Employee ID	- Occupation
- Extension number	- Number of children

In this case there would be three rows in the operational table amc_pce_party_type_data for each consumer/agent. In the event of an agent also being a consumer, there would be four rows. The amc_pce_party_type_data table is a generic table notably because of its configurable columns (float_value_01, text_value_01.....) – the business definition of these columns is stored in another table.

Amc_pce_party_type_data
Party_id
Party_type
Known_as
Float_value_01.....25
Text_value_01......25
Date_value_01......15
Rdg_value_01......25

The DataMart will present a logical representation of the party entities offering <u>one table per leaf node in the tree</u> with each leaf inheriting the attributes of the nodes above it.

Thus, in this example, there would be two tables in the DataMart each of which include the attributes of *individual* and *base party*:-

Ait_dm_p_agent
Party_id
Party_type
User Name
Password
First Name
Surname
Employee ID
Extension Number

Ait_dm_p_consumer
Party_id
Party_type
User Name
Password
First Name
Surname
Occupation
Number of children

Entities are only written to the table to which they ultimately belong: thus for a Consumer entity an entry is written to the ait_dm_p_consumer table, but not to the corresponding Base or Individual tables.

There is also a fixed table called ait_dm_party_type, which holds 'stub' information about all parties – this can be used to look up a given party in order to discover which type it is, and hence in which party table it appears. See section 2.2 for a list of all the fixed tables.

A similar 'translation' occurs for contracts with each leaf in the contract tree having its own table in the DataMart, and a similar fixed table, ait_dm_contract_product, with 'stubs' for all contract types.

In the same way, a similar 'translation' occurs for all product definitions – the 'stubs' table in this case is called ait_dm_proddftn_type.

Similarly, each type of repeating attribute, engagement action and significant event gets its own DataMart entity although these entities are not hierarchical and thus don't have any parent attributes to inherit.

2.2 Fixed tables

Fixed tables that hold operational business entity data such as engagement history are naturally exported to the DataMart. In order to aid interpretation, some of the operational entities are denormalised – thus there are some DataMart entities with more than one source table. The operational to DataMart fixed table mappings are shown in Table 1.

Table 1 - Table mappings

Function	DataMart table	Operational table
Business processes	ait_dm_business_process	amc_pce_business_process

Function	DataMart table	Operational table
Campaigns	ait_dm_cam_campaign	amc_cam_campaign
	ait_dm_cam_campaign_event	amc_cam_campaign_event
	ait_dm_cam_channel_mtrl_rate	amc_cam_channel_mtrl_rate
	ait_dm_cam_communication	amc_cam_communication
	ait_dm_cam_communication_role	amc_cam_communication_role
	ait_dm_cam_cost	amc_cam_cost
	ait_dm_cam_event_comm_method	amc_cam_event_comm_method
	ait_dm_cam_event_outcome	amc_cam_event_outcome
	ait_dm_cam_event_response	amc_cam_event_response
	ait_dm_cam_marketing_channel	amc_cam_marketing_channel
	ait_dm_cam_material	amc_cam_material
	ait_dm_cam_material_subtype	amc_cam_material_subtype
	ait_dm_cam_party_response	amc_cam_party_response
	ait_dm_cam_partycommunication	amc_cam_partycommunication
	ait_dm_cam_target_list	amc_cam_target_list
	ait_dm_cam_target_party	amc_cam_target_party
	ait_dm_cam_util_log	amc_cam_util_log
	ait_dm_sch_schedule	amc_sch_schedule
Case	ait_dm_case	amc_pce_case
Contracts/Products	ait_dm_contract_product	amc_pce_contract_data
Engagement history	ait_dm_engmt_action_party	amc_pce_engmt_action_party
		amc_pce_engagement_action
		amc_pce_engagement
	ait_dm_party_engagement	amc_pce_party_engagement
		amc_pce_engagement
	ait_dm_engagement_action	amc_pce_engagement_action
		amc_pce_engagement
Events	ait_dm_event	amc_pce_event
		amc_pce_engagement_action
		amc_pce_contract_event
Milestones	ait_dm_milestone	amc_pce_milestone
Parties	ait_dm_party_type	amc_pce_party_type_data
Product definitions	ait_dm_proddftn_type	amc_pce_proddftn_data
Reference data	ait_dm_ref_data_group	amc_rd_ref_data_group
		amc_rd_ref_data_group_depl
	ait_dm_ref_data_item	amc_rd_ref_data_item
		amc_rd_ref_data_item_depl
Relationships	ait_dm_party_event	amc_pce_party_event
	ait_dm_party_contract	amc_pce_party_contract
Workflow	ait_dm_te_task_instance	amc_te_task_instance
	ait_dm_te_task_history	amc_te_task_history
	ait_dm_te_expanded_party_rel	amc_te_expanded_party_rel
	ait_dm_te_task_exec_window	amc_te_task_exec_window
	ait_dm_te_task_exec_wind_sub	amc_te_task_exec_wind_sub

Function	DataMart table	Operational table
	ait_dm_te_task_routing ait_dm_te_task_definition ait_dm_te_task_dftn_depl	amc_te_task_routing amc_te_task_definition amc_te_task_dftn_depl

3 Process overview

You may choose to 'run' the DataMart extract in one of two modes:-

1 Full creation – the DataMart is created from scratch.

2 Incremental update – An existing DataMart is updated with the latest operational data.

Important note

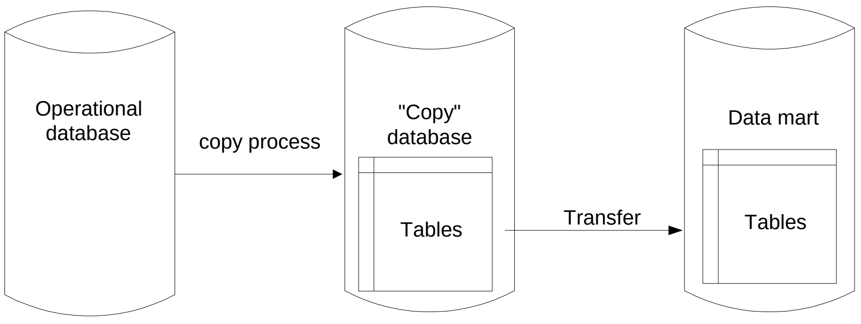
Please note that it is imperative that the DataMart process (whether operating in full or incremental mode) must run against a consistent and stable operational database image. Thus for a DataMart extract to succeed, the production database should either be taken offline, or the extract should be run against an offline copy of the live database.

Using a copy guarantees data integrity and prevents any performance degradation on the operational system. This copy may be created via any standard database method that guarantees a consistent image of the operational database. Possible methods include a simple restore or a 'continuous' method such as replication, log shipping or disk mirroring. Note that the continuous methods must be 'paused' before commencing the DataMart extract.

This document is written from the standpoint of using a copy and the term 'operational copy' is used.

3.1 Full creation

Full creation creates a DataMart from scratch using an operational copy.



The transfer processing creates the DataMart entities and transfers the data. With respect to the complex configurable entities (Parties, Contracts etc) creation process reads the Portrait Foundation meta data and creates logical representations of them in the DataMart.

The extract consists of three key phases.

1 Setup and connectivity
   - User authentication

2 SQL statement generation
   - DDL – i.e. "Create table"
   - DML – i.e. "Insert into table"
   - DDL – i.e. "Create index/constraint"

These statements are stored in tables before being executed.

3 SQL statement execution
   - Execute DDL to create tables

- Execute DML to insert data
- Execute DDL to create indexes and constraints

3.2 Incremental update

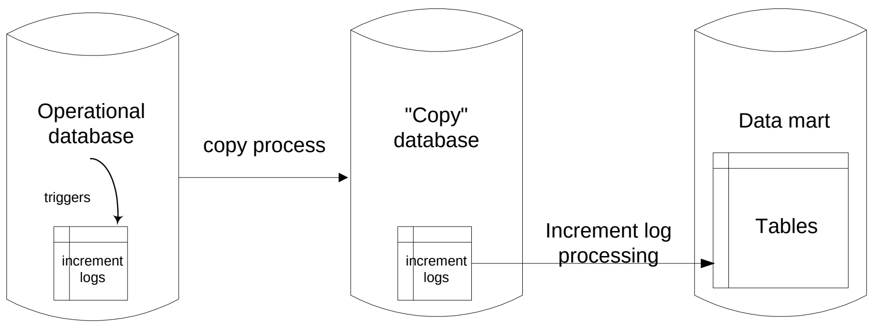
This mode only transfers operational data that has changed since you first created the DataMart or last incrementally updated it. This is clearly advantageous when the operational copy is large and little operational data has changed.

The incremental facility makes use of operational database triggers to record the fact that data has changed in the operational system.

Just as with the full creation process a copy of the operational database must be created, do not run the increment directly against the live environment

The incremental processing makes use of the incremental logs to transfer operational data changes to the DataMart.

As with the full creation, the process consists of statement generation and execution phases, with each phase having three steps.



3.2.1 Limitations

It is important to note the following limitations of the incremental update facility

1 The incremental DataMart extraction process does not cope with changes to the operational data that results from a new deployment of configuration or a product amendment. In these cases, a full DataMart creation must be run prior to any subsequent incremental extracts.

2 The incremental DataMart extraction process assumes that

- any insert operation into an operational database table increments the maximal (or decrements the minimal) primary key value
- the primary key value for a deleted row is never re-used for subsequently inserted rows.

These rules are followed by all Portrait Foundation operational database tables. You must ensure that any implementation-specific tables that you add to your operational database also abide by these rules if you require the data held by these tables to be exported incrementally to the DataMart. Similarly, data loaded via batch processes must also conform to these rules.

3 Changing the DataMart name requires a full creation of the DataMart to run – an incremental update will not run on an existing DataMart that has had its named changed. This limitation occurs because the incremental process stores and reuses fully qualified SQL statements (i.e. prefixed with the DataMart name) – thus changing the DataMart names makes these statements invalid. This limitation will typically restrict development and test

environments which would otherwise restore DataMart backups to different names to aid multi environment testing.

3.2.2 Maintenance tasks

The DataMart extract facility requires one maintenance task to be undertaken, namely a selective purge of the amc_dm_changes_log table on the operational database (not the 'operational copy').

This purge is only required in order to maximize the performance of the DataMart incremental update facility – failing to undertake the maintenance task will not result in any functional consequences however performance will decline after repeated increments.

This purge has two phases – phase one must occur before the DataMart extract runs and phase two after the extract has run.

Implementations that host the operational database and the DataMart within the same Windows domain, may choose to automate this task in a batch file or similar.

Implementations which host the operational database and DataMart on different Windows domains must undertake the task manually. Manual instructions, from which an automated task may be constructed, are listed below.

Manual steps

1 *After* the operational copy has been made, but *before* the DataMart extract is run: (whether this is to be a full build or an increment)

   a Invoke query analyzer against the operational copy connecting with any account that has sysadmin privileges (e.g. sa)

   b Run this SQL : select max(changes_log_id) from amc_dm_changes_log

This will return the top change id which will be part of the extract

2 *After* the DataMart extract has been successfully run:

   a Invoke query analyzer against the operational database (not the operational copy) connecting with any account that has sysadmin privileges (e.g. sa)

   b Run this SQL: exec p_amc_dm_purge_live_changes_log @p_max_changes_log_id

where @p_max_changes_log_id is the value returned in step 1b

4 Installation

The DataMart Creation Utility is available for install from the Portrait Foundation distribution media.

Please consult the associated document *Portrait_Installation_Guide* for full details of how to install the DataMart creation utility. In brief, a custom install should be used and only the DataMart install needs to be selected.

5 Considerations

Before creating a DataMart, the user must consider the following important pieces of information.

5.1 Prerequisites

This user guide makes the following assumptions with regard to the operating environment prior to running the DataMart Creation Utility:

- The client device, from where you intend to run the utility, has the SQL Server Client and the DataMart Creation Utility installed.
- An online Portrait Foundation database and a DataMart already exist. The DataMart may be empty or may have been populated previously. The utility simply requires a database shell to work with.
- The database and DataMart both sit on a SQL Server 2005 platform or higher. They must also be on the same instance of SQL Server.
- The Windows account running the utility or SQL Server user specified in the 'Create DataMart' dialog must be the database owner of the relevant database or be a member of the SQL Server 'System Administrator' server role.
- The names of the database and the DataMart do not exceed 39 characters in length. This is due to restrictions imposed by the utility to ensure control over the length of generated SQL commands.

5.2 Backup precautions

Portrait strongly recommends that before running this utility, the user ensures that the DataMart has been backed up. In the event of an aborted attempt to rebuild the DataMart, there is no way of undoing the changes that the utility may have made. The only way to revert to the healthy DataMart is to restore the backup taken at this point. Alternatively you may fix the problems and regenerate the DataMart.

The DataMart does not affect any changes to the business data in the online database, so there is no need to take a backup of the online database at this point. However, it should be noted that if changes occur to the business data during the process of loading the DataMart, this may cause errors or render the resulting DataMart inconsistent. It is therefore recommended to perform DataMart extracts against an ***offline*** backup of the online database. It is in any case assumed that the customer has a general backup strategy for the maintenance and security of the live environment as a matter of good practice.

5.3 Duration

5.3.1 Full creation

A full DataMart creation can involve a large amount of data extraction and manipulation, depending on the size of the Portrait Foundation operational database. Depending on the environment, this can therefore be a long running operation and also generate a large amount of network traffic.

One should thus consider how long the utility takes to complete its routine from start to finish – users should allow a suitable time window to create the DataMart. It is possible that many millions of records will be present in the operational system and that consequently many millions of records will be transferred and transformed via the DataMart creation process. Example timings are 4 hours using a 4-core server acting on an operational copy with 5 million parties, 10 million engagements and 15 million repeating attributes. Naturally these timings will vary depending on the database and server size.

By using the Settings dialog to select which entities are exported to the DataMart, these timings can be reduced. In particular, export of Task types can be particular time-consuming, so a great deal of time can be saved by excluding these from the export.

5.3.2 Incremental update

The incremental update facility will normally be substantially quicker than a full create as it only considers data changes since the last increment/full create. That said, the duration will vary considerably depending on the amount of data changed.

Example timings have been found to be 1 hour on a 4-core server with 30,000 data amendments to reflect in the DataMart and approximately 75,000 new rows to insert into the DataMart.

Note that you cannot change the export settings for an incremental update – the settings from when the DataMart was originally created will be used.

5.3.3 Server topology

The use of linked servers is no longer supported from version 4.4 Update 4. This is primarily because of the substantial performance issues associated with running SQL across databases servers. For this reason, we only recommend that the operational copy and DataMart databases are on the same SQL Server instance.

This can be achieved through many duplication or mirroring approaches such as hardware mirroring or SQL log shipping. It is the customer’s responsibility to ensure that the database environment and architecture is set up correctly before starting the DataMart creation utility.

If, after co-locating the operational copy and DataMart, performance problems still persist, consider the performance diagnosis steps as described in section A.2.1.

6 The creation utility

The DataMart creation utility may be run interactively as a dialog, or in batch mode through a script or the command line interface. These options operate the same underlying code and logic – only the user interface changes.

This section concentrates on the interactive version of the utility as it allows the easiest feature explanation. Section 6.2.discusses the equivalent batch mode features.

6.1 Running interactively

6.1.1 Starting the utility

You can start the utility in one of two ways:

- By double-clicking on the **DatamartCreationU.exe** file in the **<C:\Program Files\PST\Portrait Foundation\common\bin>** folder[1].

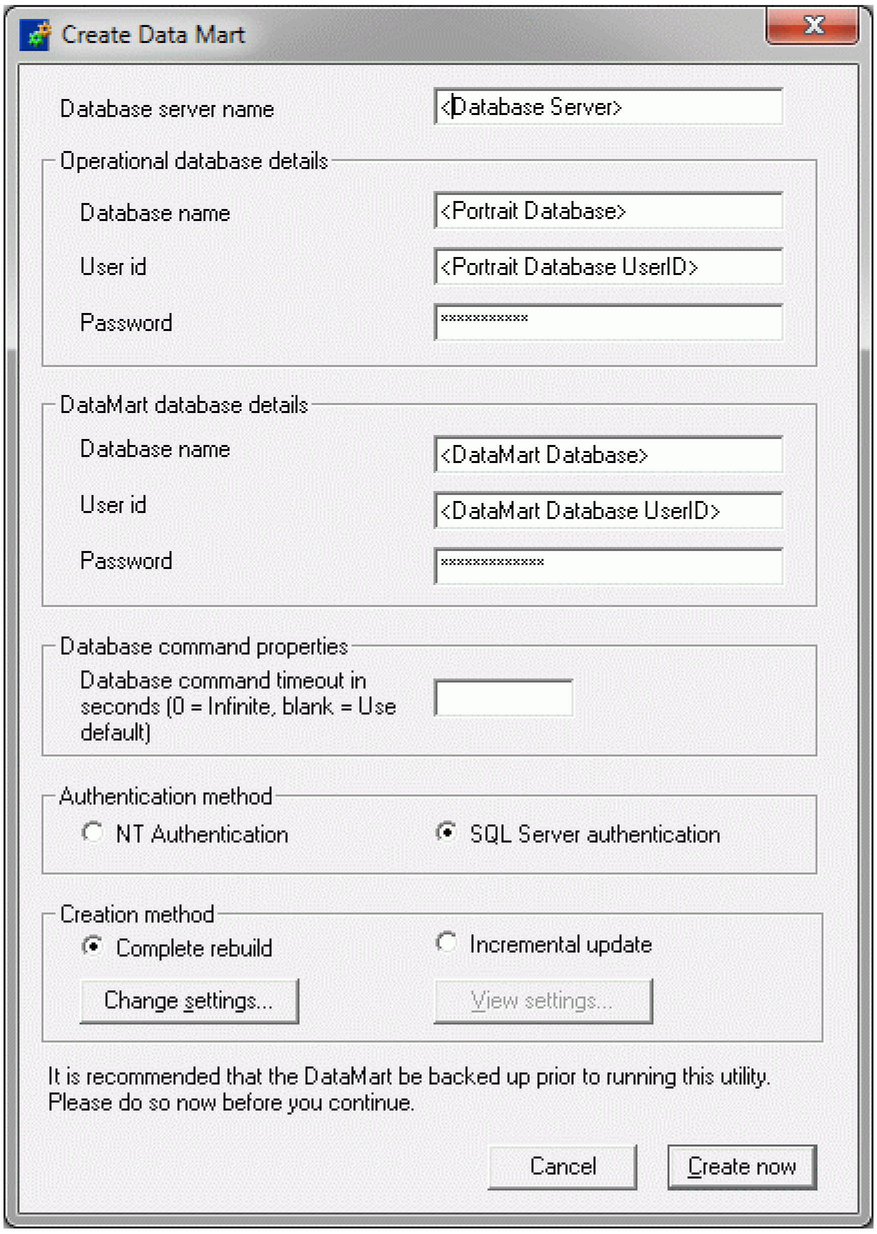
By selecting the **DataMart Creation Utility** option from **Start \All Programs\Portrait Foundation\[*MyPortrait*]\System Tools.** Where [*MyPortrait*] is the name of your Portrait system.

6.1.2 Log on screen

The prompts that appear on the log on screen are determined by the authentication method chosen (Windows or SQL Server). This information is used to enable connection to the Operational database and the DataMart.

[1] For a 64-bit server this path will be <C:\Program Files (x86)\PST\Portrait Foundation\common\bin> by default.

Figure 1 – Create DataMart dialog using SQL Server authentication


Create Data Mart

Database server name	<Database Server>

Operational database details

Database name	<Portrait Database>
User id	<Portrait Database UserID>
Password	************

DataMart database details

Database name	<DataMart Database>
User id	<DataMart Database UserID>
Password	**************

Database command properties

Database command timeout in seconds (0 = Infinite, blank = Use default)

Authentication method

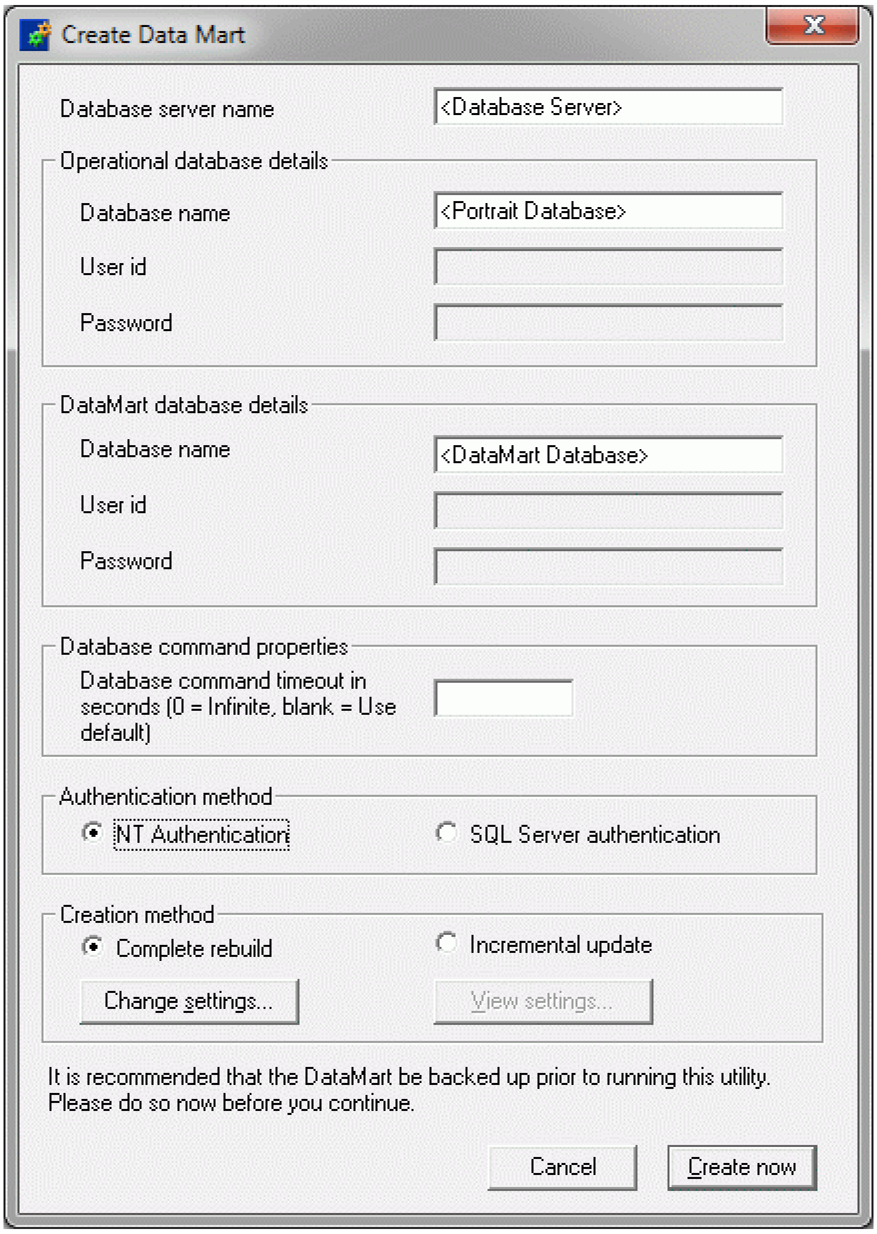
NT Authentication [✓ SQL Server authentication]

Creation method

[✓ Complete rebuild] Incremental update

It is recommended that the DataMart be backed up prior to running this utility. Please do so now before you continue.

Figure 2 – Create DataMart dialog using NT authentication



Database server name: <Database Server>

Operational database details
- Database name: <Portrait Database>
- User id:
- Password:

DataMart database details
- Database name: <DataMart Database>
- User id:
- Password:

Database command properties
- Database command timeout in seconds (0 = Infinite, blank = Use default):

Authentication method: [✓ NT Authentication] SQL Server authentication

Creation method: [✓ Complete rebuild] Incremental update

It is recommended that the DataMart be backed up prior to running this utility. Please do so now before you continue.

In the examples in Figure 1 and Figure 2, the strings in angle brackets would be replaced by the relevant names of the database server, the operational database, the DataMart, and their respective user ids and passwords. Note that if NT Authentication is selected, all userid and password entry fields will be greyed out.

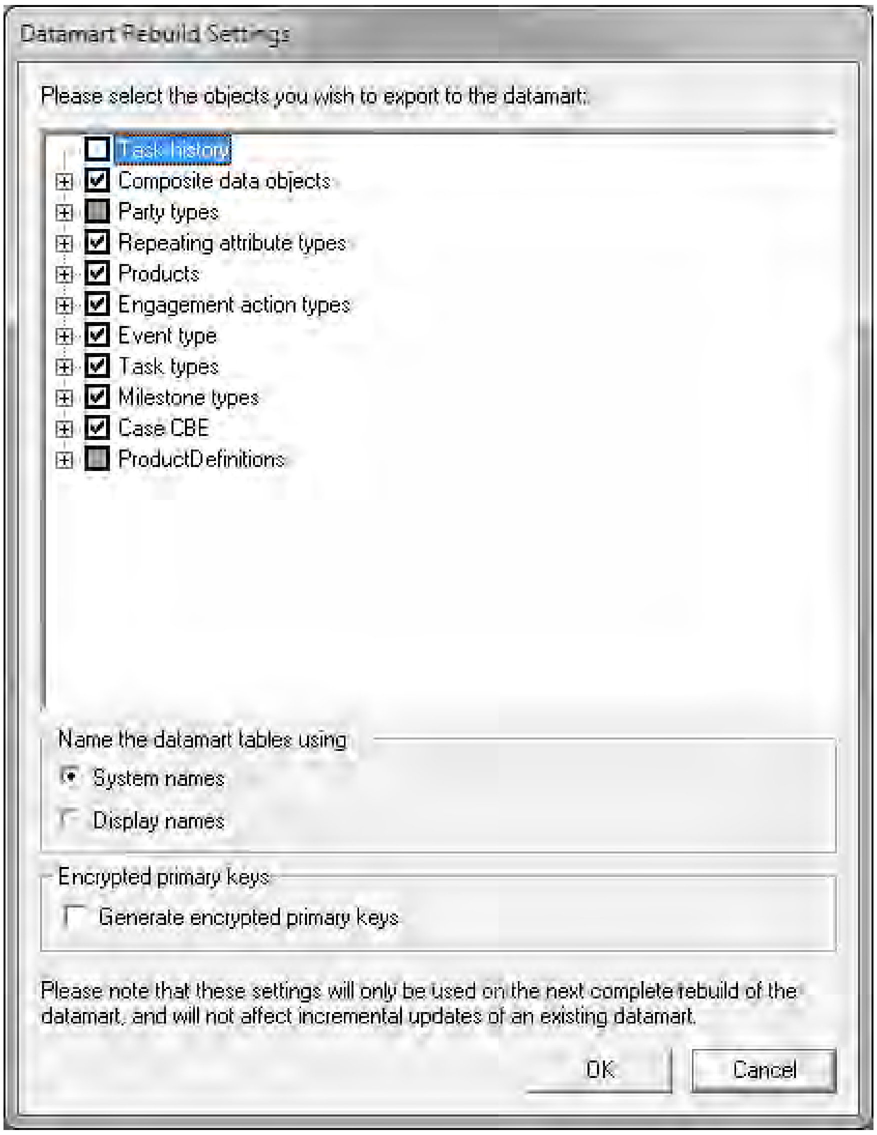
The user id must have DBO privileges on both the operational copy and the DataMart.

The creation method option determines whether the utility attempts to perform an incremental update or a complete rebuild. For a complete rebuild, the existing contents of the DataMart are deleted, and all of the data in the operational copy is exported afresh. In incremental mode, however, it will attempt to copy only those records that have changed or been added, and will delete records from the DataMart that have been deleted from the operational copy. Naturally, for this to work, the DataMart database must previously have been populated from the same operational database. Errors will occur if an empty DataMart database is used with this option, or one that has been populated from a different operational database. Note that it is not possible to perform an incremental update if the configuration data has been redeployed since the last DataMart load.

6.1.3 Settings dialog

Pressing the **Change settings…** button allows you to change the settings used to create the DataMart. It brings up the following dialog:

Figure 3 – DataMart Rebuild Settings dialog



Datamart Rebuild Settings

Please select the objects you wish to export to the datamart:

- [ ] Task history
- [✓] Composite data objects
- [■] Party types
- [✓] Repeating attribute types
- [✓] Products
- [✓] Engagement action types
- [✓] Event type
- [✓] Task types
- [✓] Milestone types
- [✓] Case CBE
- [■] ProductDefinitions

Name the datamart tables using

[● System names] ○ Display names

Encrypted primary keys

- [ ] Generate encrypted primary keys

Please note that these settings will only be used on the next complete rebuild of the datamart, and will not affect incremental updates of an existing datamart.

Here you can select which entities from the operational database should be exported to the DataMart. In order to reduce the time taken by the extraction process, it is advisable only to export those entities that you require. In particular the export of Task history, Composite data object types and Task types can be especially time consuming, so if they are not required in the DataMart it is advisable to deselect them.

You can also choose whether to use the system names or the display names of your entities in constructing the DataMart table and column names. We advise you to use system names, as this means that the DataMart structure will remain unchanged if you change the display names of your entities.

There is also the option to encrypt the primary keys so that they are not immediately traceable back to the operational data.

Press the OK button to return to the previous screen. This saves the settings you have chosen back to the operational database, where they will be used as the defaults the next time you create a new DataMart. Note that if you are working from a read-only copy of the operational database, this save operation will fail and you will see an error message. This indicates that the settings you have chosen will not be persisted for when you next create a DataMart, but that it is still safe to continue with the DataMart creation process. The settings you have just chosen will be used.

If you are updating an existing DataMart instead of creating a new one, only the **View settings…** button is enabled. This allows you to view the settings that were used when the DataMart was created, but you cannot change them.

Having entered the necessary information, click **Create now.**

6.1.4 Create now button

1 The DataMart framework will check the operational database to ensure that the DataMart functionality has been enabled. It does this by checking that all of the standard triggers that populate the operational database's `amc_dm_changes_log table` are enabled. They can become disabled in one of three ways:

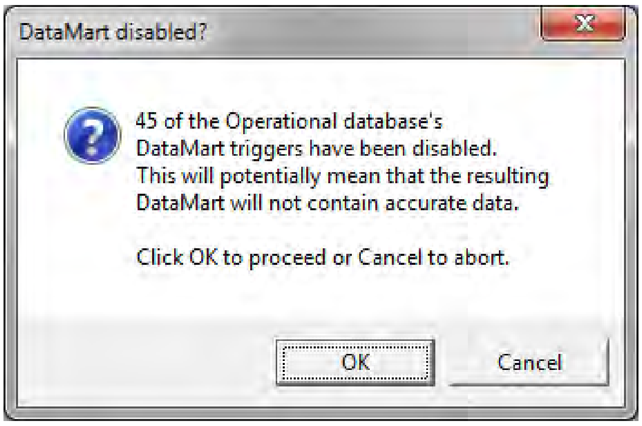
   a When the operation database was created, the 'Include DataMart functionality' checkbox was unchecked, or

   b The utility stored procedure `p_amc_utl_disable_dm_changes_log` has been run passing the `@p_disable_it` parameter as either null or 1, or

   c By a DBA manually disabling one or more of the (approx.) 45 triggers in the operational database that help to populate the `amc_dm_changes_log table`.

   In this event, the following message will be shown:

Figure 4 –DataMart disabled message



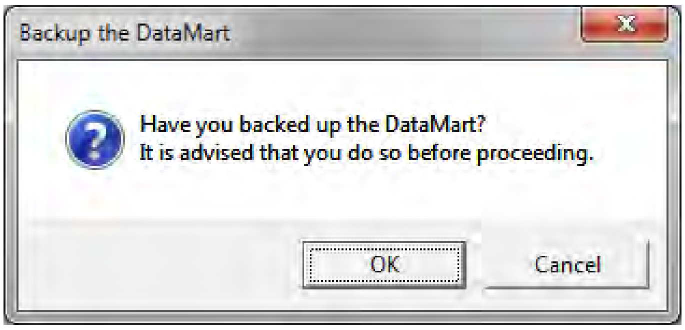
45 of the Operational database's DataMart triggers have been disabled.
This will potentially mean that the resulting DataMart will not contain accurate data.

Click OK to proceed or Cancel to abort.

   You are strongly advised to investigate the issue before continuing with the DataMart creation, as otherwise, the outcome may be unpredictable and not representative of the actual data in the operational database. Contact Portrait Support for more details.

2 You are asked to confirm that the DataMart has been backed up as a precaution.

Figure 5 – Back up The DataMart message

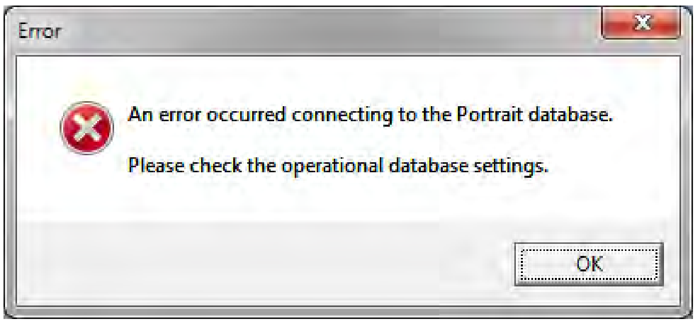


Backup the DataMart

Have you backed up the DataMart?
It is advised that you do so before proceeding.

3 The utility checks for connectivity to both databases. If unsuccessful, you will see a message box similar to the ones illustrated in Figure 6:

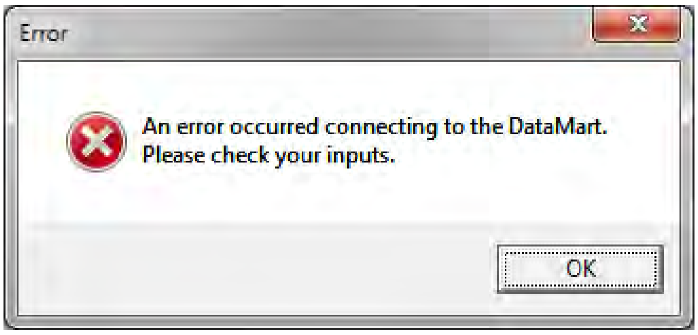
Figure 6 – Connectivity error messages



Error

An error occurred connecting to the Portrait database.

Please check the operational database settings.



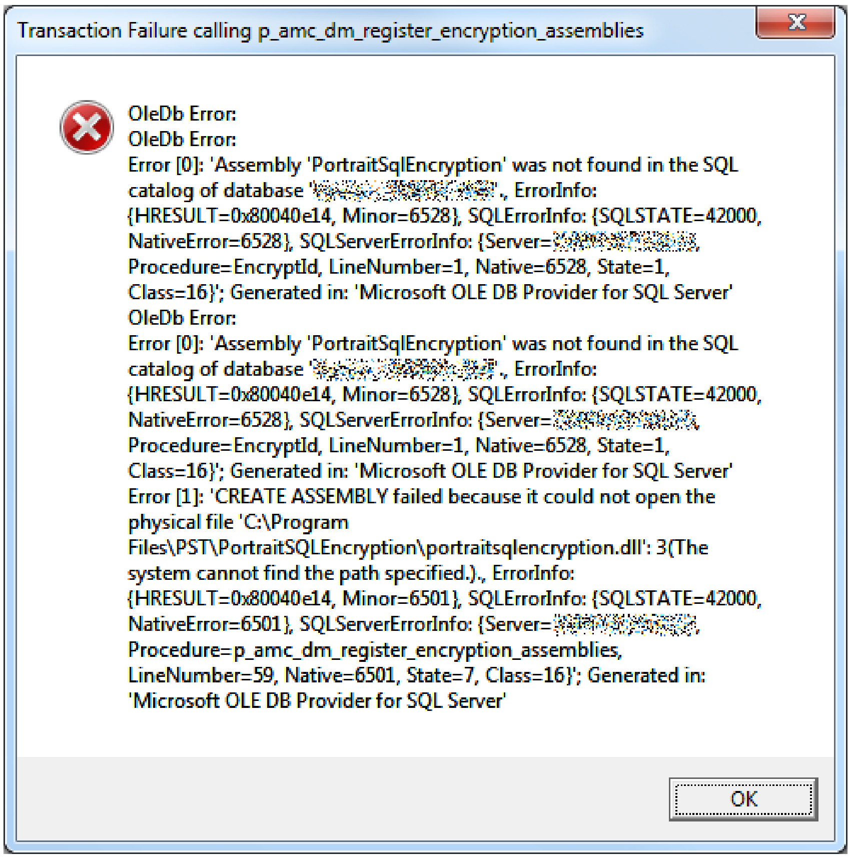
Error

An error occurred connecting to the DataMart.
Please check your inputs.

An error such as these probably indicates misspelled user inputs. If this is not the case, it is possible that the user ids input do not have system administrator privileges, the operational database is not set up correctly or does not contain the necessary stored procedures. If so, contact Portrait Support.

4 If you have chosen the option to generate encrypted primary keys and have not installed support for that feature on the database server the following error message will be displayed. Either uncheck this feature in the settings dialog or install the components on the database server.

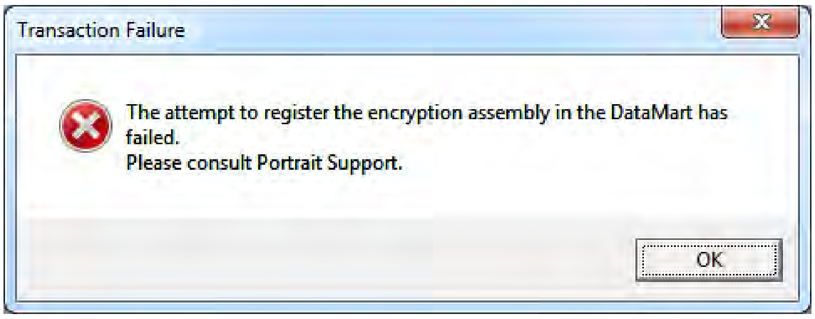
Figure 7 – Support for primary key encryption has not been installed on the SQL server



Transaction Failure calling p_amc_dm_register_encryption_assemblies

```
OleDb Error:
OleDb Error:
Error [0]: 'Assembly 'PortraitSqlEncryption' was not found in the SQL
catalog of database '[redacted]'., ErrorInfo:
{HRESULT=0x80040e14, Minor=6528}, SQLErrorInfo: {SQLSTATE=42000,
NativeError=6528}, SQLServerErrorInfo: {Server=[redacted],
Procedure=EncryptId, LineNumber=1, Native=6528, State=1,
Class=16}'; Generated in: 'Microsoft OLE DB Provider for SQL Server'
OleDb Error:
Error [0]: 'Assembly 'PortraitSqlEncryption' was not found in the SQL
catalog of database '[redacted]'., ErrorInfo:
{HRESULT=0x80040e14, Minor=6528}, SQLErrorInfo: {SQLSTATE=42000,
NativeError=6528}, SQLServerErrorInfo: {Server=[redacted],
Procedure=EncryptId, LineNumber=1, Native=6528, State=1,
Class=16}'; Generated in: 'Microsoft OLE DB Provider for SQL Server'
Error [1]: 'CREATE ASSEMBLY failed because it could not open the
physical file 'C:\Program
Files\PST\PortraitSQLEncryption\portraitsqlencryption.dll': 3(The
system cannot find the path specified.)., ErrorInfo:
{HRESULT=0x80040e14, Minor=6501}, SQLErrorInfo: {SQLSTATE=42000,
NativeError=6501}, SQLServerErrorInfo: {Server=[redacted],
Procedure=p_amc_dm_register_encryption_assemblies,
LineNumber=59, Native=6501, State=7, Class=16}'; Generated in:
'Microsoft OLE DB Provider for SQL Server'
```

This is followed by:



Transaction Failure

The attempt to register the encryption assembly in the DataMart has failed.
Please consult Portrait Support.

Unchecking the option to generate encrypted ids will allow the DataMart creation to proceed, however the encrypted id columns will be NULL. To fix this, install the DataMart encryption components on the database server by following the guidelines in the *DataMart_encryption_installation* document.

5 The utility interprets the Portrait Foundation metadata and generate a series of SQL commands designed to create the structure of the DataMart, that is, its tables and constraints. These commands will be stored in the DataMart database in tables called `amc_dm_table_sql` and `amc_dm_post_data_sql`.

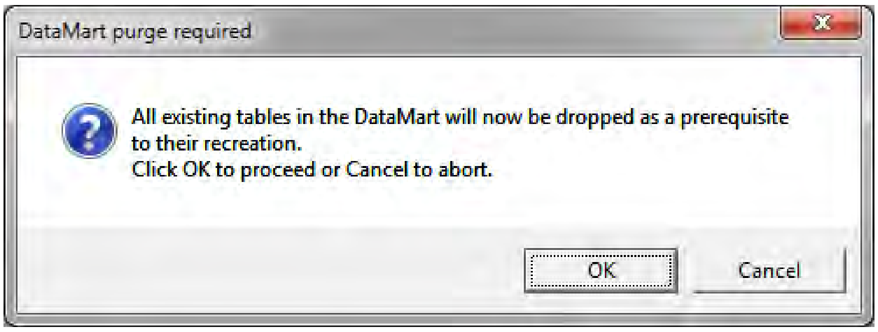
6 Generate a further set of SQL statements ready to perform the data inserts necessary to populate the DataMart with client and product data. These commands will be stored in the DataMart database in a table called `amc_dm_data_sql`.

The last two steps should not result in any errors. An error at this stage probably represents a fundamental problem and you are advised to abort the process before any changes have been made to the DataMart by clicking **Cancel** on the initial **Create DataMart** dialog.

6.1.5 (Re)creating the DataMart

1 Assuming that the SQL commands were generated without error, the dialog illustrated in Figure 8 is displayed, but only in the case of a complete rebuild. Otherwise no purge is required and this step is omitted.

Figure 8 - DataMart purge required

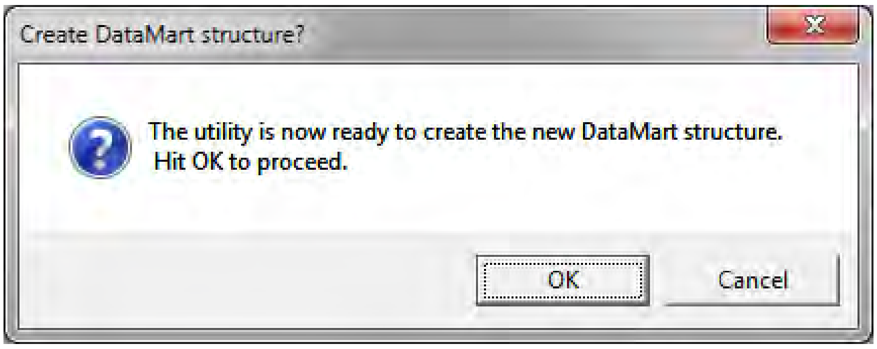

DataMart purge required

All existing tables in the DataMart will now be dropped as a prerequisite to their recreation.
Click OK to proceed or Cancel to abort.

2 Click **OK** to continue, or **Cancel** to abort the process.

3 The utility is now ready to upgrade/create the DataMart structure, and will ask for confirmation similar to that shown in Figure 9.

Figure 9 - Upgrade DataMart structure message

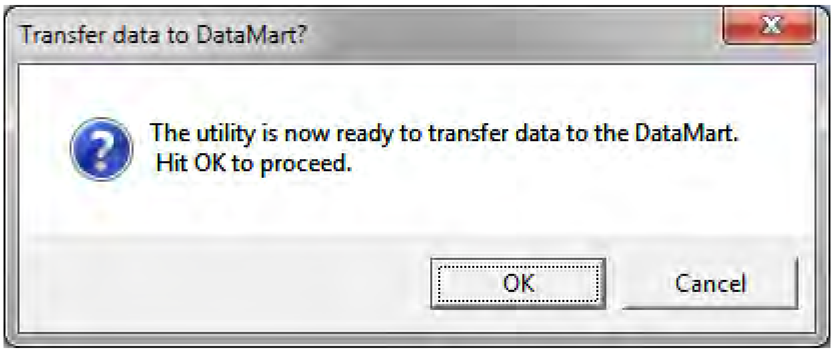

Create DataMart structure?

The utility is now ready to create the new DataMart structure.
Hit OK to proceed.

6.1.6 Populating the DataMart

1 Once the structure has been created, the next step is to populate it with data. Click **OK** to continue.

Figure 10 - Transfer data to DataMart dialog


Transfer data to DataMart?

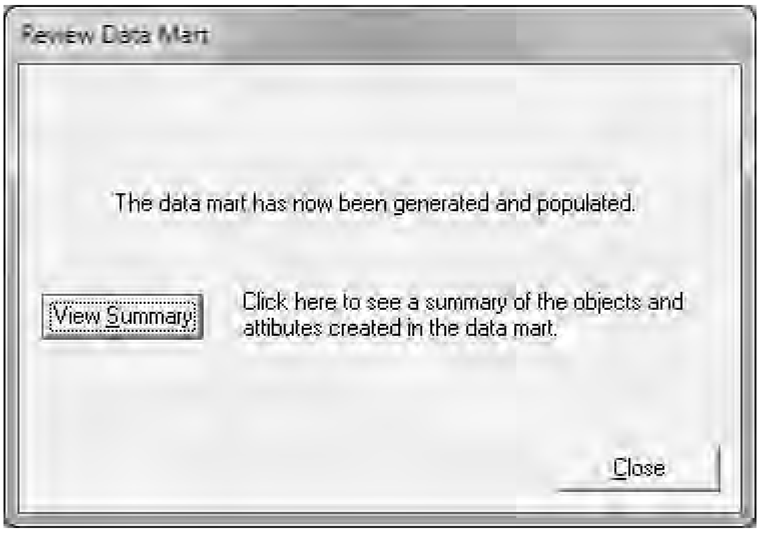
The utility is now ready to transfer data to the DataMart.
Hit OK to proceed.

2 It is possible that errors could occur when executing this command. For a discussion of how to handle these refer to Section Appendix A on page 41.

6.1.7 Review DataMart

Once the DataMart has been loaded, the **Review DataMart** dialog is displayed.

Figure 11 – Review DataMart dialog



The data mart has now been generated and populated.

Click here to see a summary of the objects and attibutes created in the data mart.

At present, there is only one type of review that can be undertaken and that is to view a summary of the objects created in the DataMart. Future versions of the tool may provide further options at this point.

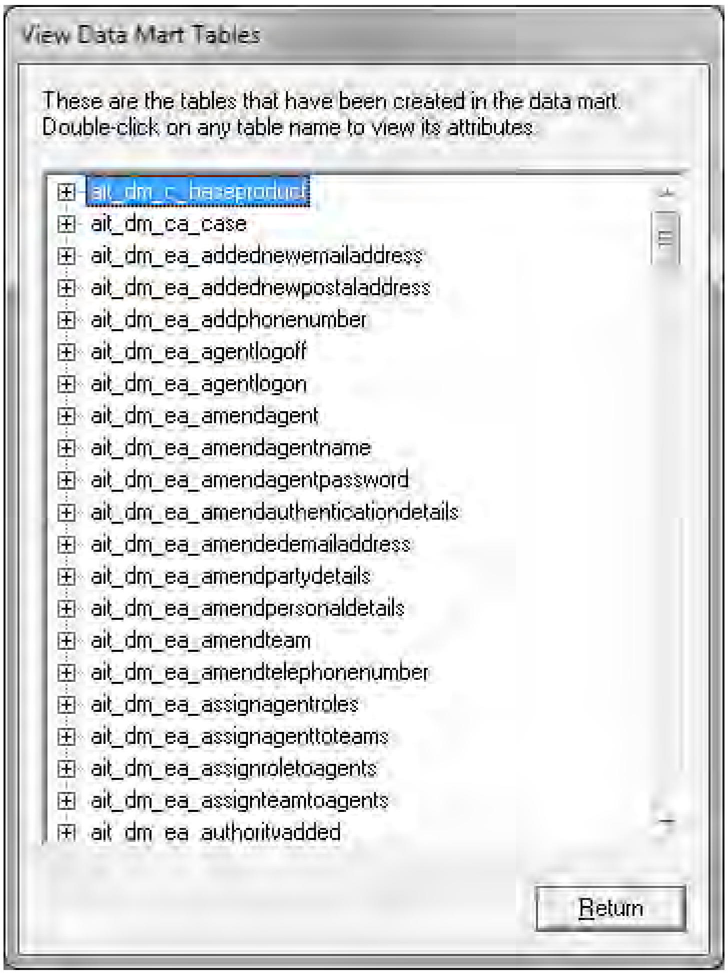
Click **View Summary** for the summary display illustrated in Figure 12. Click **End** to exit the utility.

6.1.8 DataMart objects summary

The **View DataMart Tables** dialog provides a tree diagram in which all the tables that have been created in the DataMart can be viewed at the top level. By expanding the node of a given table, you can examine the details of the columns within the table.

Note that this dialog does not list the extra **Reference Data Resolution Views** that are generated on top of the DataMart tables. For every table created in the DataMart, there is a corresponding view with the same name, but prefixed with "v_". This view will contain all the same columns, with the addition of two extra columns for each pair of reference data columns stored in the underlying table. The columns in the underlying table contain the reference data item and group Ids that identify a particular item of reference data, but these can be hard to decipher; the view joins to the reference data item table and resolve this ids to a display name and a system name.

Figure 12 – View DataMart Tables dialog


View Data Mart Tables

These are the tables that have been created in the data mart.
Double-click on any table name to view its attributes.

- ait_dm_c_baseproduct
- ait_dm_ca_case
- ait_dm_ea_addednewemailaddress
- ait_dm_ea_addednewpostaladdress
- ait_dm_ea_addphonenumber
- ait_dm_ea_agentlogoff
- ait_dm_ea_agentlogon
- ait_dm_ea_amendagent
- ait_dm_ea_amendagentname
- ait_dm_ea_amendagentpassword
- ait_dm_ea_amendauthenticationdetails
- ait_dm_ea_amendedemailaddress
- ait_dm_ea_amendpartydetails
- ait_dm_ea_amendpersonaldetails
- ait_dm_ea_amendteam
- ait_dm_ea_amendtelephonenumber
- ait_dm_ea_assignagentroles
- ait_dm_ea_assignagenttoteams
- ait_dm_ea_assignroletoagents
- ait_dm_ea_assignteamtoagents
- ait_dm_ea_authorityadded

Click **Return** to return to the **Review DataMart** dialog and exit the utility.

6.2 Running in batch mode

When running the generated SQL commands, either for the structural changes or for the data insertion, errors may occur. An error at this stage generally indicates that data has been incorrectly deployed through the Configuration Suite or that there is some other kind of data corruption.

For this reason it is recommended that the interactive version of the utility is used in development environments to prove the Portrait Foundation configuration and highlight any errors or potential conflicts before the batch version is run.

It is also not possible to access the DataMart rebuild settings dialog in batch mode, so the set of entities exported will be determined by the last time the process was run interactively on that operational database. It only necessary to visit the Rebuild settings dialog using the interactive utility, choose the entities to export and close it by pressing OK in order to save these settings – you do not have to progress to actually creating a DataMart.

6.2.1 Starting the utility

The batch **DMCreateU.exe** utility can be started in one of three ways:

- From the command line interface.
- By executing a **.bat** file which contains the required invocation parameters.
- Any scheduling application capable of running an **.exe** from the command line and passing it parameters.

6.2.2 Input parameters

The input parameters are determined by the authentication method chosen (Windows or SQL Server). The following named parameters delimited by **/** are passed to the utility:

/dbserver=	The server on which the requested operational copy resides.
/db=	The name of the Portrait operational copy to be used.
/dbuid=	Only valid when SQL Server authentication is used. A valid SQLServer user id that has all the necessary database privileges on the operational copy.
/dbpw=	Only valid when SQL Server authentication is used. A valid password for the user id specified above.
/dm=	The name of the Portrait DataMart to be used—this must already exist.
/dmuid=	Only valid when SQL Server authentication is used. A valid SQL Server user id that has all the necessary database privileges on the DataMart.
/dmpw=	Only valid when SQL Server authentication is used. A valid password for the user id.
/ntuid	Only valid when Windows authentication is to be used When this parameter is present **dbuid**, **dbpw, dmuid** and **dmpw** are ignored. This parameter does not require a value to be given, it simply instructs the utility to use the currently logged on user.
/reportfile=	A file name and location where a list of the generated tables and their columns will be written.
/eventlog	Output runtime errors to the Windows event log rather than the console.
/update	Perform an incremental update instead of a complete rebuild.
/generateonly	Stops the DataMart process after the SQL statements have been generated, before they are executed.
/timeout=	An opportunity to override the default SQL query timeout with a new value (in seconds).
/displaynames	The utility will use the display names of entities when generating the names of tables and columns, instead of system names. Note that this means the table names can change if you change the display names of the entities. The default is to use system names.
/encryptids	Flag to inform the system to encrypt the primary keys.

/keepindexes	Flag to prevent the system from dropping the DataMart tables' indexes during an Update. This can sometimes prove to be quicker when adding updates to a very large DataMart. You should test whether this improves your Update performance before using it.

Sample DMCREATEU parameter string using SQL Server authentication:

```
C:\Program Files\PST\Portrait Foundation\common\bin> dmcreateu /dbserver=<Portrait
Database Server> /db=<Portrait Database> /dbuid=<Portrait User Id> /dbpw=<Password for
User Id> /dm=<DataMart Database> /dmuid=<DataMart UserId> /dmpw=<Password for User Id>
/reportfile=<Database Report File> /eventlog
```

Sample DMCREATEU parameter string using NT authentication

```
C:\Program Files\PST\Portrait Foundation\common\bin > dmcreateu /dbserver=<Portrait
Database Server> /db=<Portrait Database> /ntuid /dm=<DataMart Database>
/reportfile=<Database Report File> /eventlog
```

The strings in angle brackets must be replaced by the relevant names of the database server, operational database, DataMart and user ids and passwords of their respective databases.

6.2.3 Batch process flow

The batch utility runs the same stored procedures as invoked by the interactive version and the process flow is almost identical, the only difference being the removal of the user prompts.

1. Check for connectivity on both databases.
2. Interpret the Portrait Foundation metadata and generate a series of SQL commands designed to create the structure of the DataMart.
3. Generate a further set of SQL statements ready to perform the data inserts necessary to populate the DataMart.
4. Assuming that the SQL commands were generated without error, the DataMart structure will be (re)created.
5. Once the structure has been created, the next step is to populate it with data.

6.3 Conflict resolution

After the DataMart has been created and populated, a stored procedure will run that will highlight any structural conflicts that occurred during the generation process.

These will invariably concern conflicting column names that have been caused by similarly named attributes within an entity's hierarchy.

The conflicting columns will be resolved by using the defaults supplied by the stored procedure and a report will be produced detailing each conflict and the name used to resolve it. They are also listed in the table `ait_sum_column_conflict` in the DataMart. If the default column names are not acceptable, they may be resolved by amending the Portrait Foundation configuration so that no conflict occurs.

6.4 NT Authentication

When NT Authentication is selected, the utility makes use of the security features built into windows to log on to SQL Server. When the utility attempts to connect to the database server, SQL Server will automatically validate the user account under which the utility is being run. It will then check whether this account has been granted login access and what permissions it has. All of this is managed by SQL Server and the operating system at a low level—the utility itself is not involved in the authentication process at all, and has no knowledge of the account under which it is running.

7 Extension framework

7.1 Overview

The shipped utility extracts information from the tables in the Portrait Foundation database schema; if implementation teams add extra tables to the Portrait Foundation database and wish to export this data to the DataMart, the utility needs to be extended to cope with these extra tables.

The DataMart utility provides some simple "hooks" to allow implementations to plug their own tables into the generic framework. Note that in some cases, the hooks require specific object names to be used. When 'hooked' the implementation objects will be used in both full creation and incremental update.

Before reading the following material, the reader should familiarise themselves with the basic process of creating a Portrait Foundation DataMart as described in section 2.

It is important to note that the DataMart extract process has two phases, statement generation and statement execution. The execution phase is generic and does not change regardless of whether or not implementation objects are hooked in.

Consequently the extension framework focuses on the first phase, statement generation. The creation of the correct DDL and DML statements is the key to including implementation objects in the DataMart.

7.2 Getting started

This section discusses the various considerations that must be addressed in order to extend the DataMart by way of implementation specific tables. Details of the technical components that support the principals involved are listed in section 7.3.

The basic process can be split as follows

1. Determine the operational to DataMart data mapping
2. Create operational database triggers to record data amendments and make sure that they are not disabled
3. Generate DDL statements to create the DataMart tables
4. Generate DML statements to transfer data into the DataMart.
5. Generate control statements to maintain some DataMart creation control data.

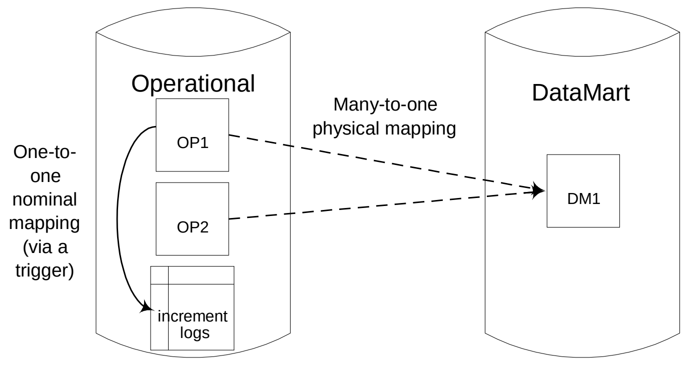
7.2.1 Data mapping

Before undertaking any technical work, some logical business mapping is required. It is necessary to consider which operational tables will serve as the data source for which implementation tables.

Though a DataMart table can be sourced from the combined content of several operational tables, each DataMart table must have a single source table nominated as the owning table that indicates when new content is available for export. This is the source table that will have the trigger for update/delete events and also be used by the procedure to recognise newly inserted rows. All necessary data from the combined source tables will be exported, but it is the nominated table that acts as the master indicator. This one-to-one nominal mapping is required in order to simplify what would otherwise be vastly more complex incremental processing.

For example (see Figure 13) consider a DataMart table, DM1. It has two operational physical source tables, OP1 and OP2, which are joined to form the data source for DM1. In this case only one of the two operational tables (say OP1) is studied for data changes and thus only changes to OP1 can trigger the need to increment DM1. OP2 changing in isolation will not cause an increment although OP2's data is reloaded into the DataMart when OP1 triggers the need to increment DM1.

Figure 13 – Nominal mapping example


Operational
OP1
OP2
increment logs
One-to-one nominal mapping (via a trigger)
Many-to-one physical mapping
DataMart
DM1

Thus before undertaking any code, one must consider what the nominal mappings are. Note that this feature is irrelevant if the operational to DataMart physical tabular mappings are one-to-one.

7.2.2 Triggers

With the data mappings determined, it is necessary to create operational database triggers for each nominal source table – these triggers will log data amendments in the operational system which then serves as a key input into the incremental update process. Please consult section 7.3.7 for technical details on these triggers.

7.2.3 Data Definition Language (DDL)

Clearly a key stage is the generation of suitable DDL statements to create the appropriate implementation specific DataMart objects. Note that implementations need only be concerned with the generation of these statements – their execution occurs automatically as part of the extract utility (refer to section 7.1).

The DDL statements need to be generated in a specific stored procedure (`p_imp_dm_create_imp_tables`) – once generated, tabular statements will be stored separately from index and constraint statements. Please consult section 7.3.9 for further details.

7.2.4 Data Manipulation Language (DML)

In addition to the DDL, DML statements are required to transfer the operational data into the DataMart. As with the DDL, implementations need only be concerned with the generation of these statements – their execution occurs automatically as part of the extract utility (see section 7.1).

The DML statements need to be generated in a specific stored procedure (`p_imp_dm_transfer_imp_data`) – both deletion and insertion statements will be required. The same DML statements, albeit with different parameters, will be used in both a full creation and incremental update. Please consult section 7.3.10 for further details.

7.2.5 Control statements

In addition to the core DDL and DML statements it is also necessary to maintain some simple yet essential DataMart creation control data. This control data is particularly important for effective and accurate incremental updates. SQL statements to maintain this control data are located among both the DDL and

DML statements. Thus both sections 7.3.9 and 7.3.10 have some reference to this control information.

7.3 Components

Several key components form the backbone of the extension framework. Some of these components are packaged as part of the product, some must be created by the implementation team. The components are summarised in Table 2 and are discussed in more detail in the sections that follow.

Table 2 - Extension framework components

Component	Specified name	Description	Location	Product or implementation	Used in full creation / incremental
Logging table	amc_dm_changes_log	Table used by triggers to store row update or deletion events	Operational database, copied to DataMart	Product	Incremental only
Increment control table	ait_ctl_increment_log	Table that lists the increments that have been run and their details	DataMart	Product	Incremental only
Table control table	ait_ctl_table_log	Table that lists the max and min primary key values of each operational table that are present in the DataMart	DataMart	Product	Both
Table summary	ait_sum_new_tables	Table that lists the tables that have been created in the DataMart	DataMart	Product	Both
Column summary	ait_sum_new_columns	Table that lists the columns of the tables that have been created in the DataMart	DataMart	Product	Both
Column conflicts	ait_sum_column_conflict	Table that lists any column conflicts, and the names they were given to resolve them.	DataMart	Product	Both
Triggers	t_imp_upd/del_<table_name>	Operational database triggers that log a row update or deletion	Operational database	Implementation	Incremental only
SQL Storage tables	amc_dm_table_sql amc_dm_data_sql amc_dm_post_data_sql	Stores 'Create table' like DDL Stores DML used in data transfer Stores 'Create constraint/index' like DDL	DataMart	Product	Both
Generate implementation DDL procedure	p_imp_dm_create_imp_tables	A procedure that generates the DDL to create implementation objects in the DataMart	Operational database	Implementation	Full only
Generate implementation DML procedure	p_imp_dm_transfer_imp_data	A procedure that generates the DML to transfer implementation object data to the DataMart	Operational database	Implementation	Both
CBE options table	amc_dm_opt_cbe_options	Table that stores the options chosen as to which configurable business entities (CBEs) should be exported to the DataMart	Operational database, copied to the DataMart	Product	Both
General options table	amc_dm_opt_gen_options	Table which stores other options used in creating the DataMart	Operational database, copied to the DataMart	Product	Both

7.3.1 amc_dm_changes_log

This table is used to store the changes that occur in the operational database and is seeded by operational database triggers (see section 7.3.7). Its structure is described in Table 3

Table 3 - amc_dm_changes_log table

Column	Usage
Changes_log_id	Primary key - identity column
Table_name	Name of table which trigger belongs to
Operation	Was a row updated ('UPDATE') or deleted ('DELETE'). Note inserts are not recorded by triggers and thus don't appear in this table.
Primary_key	The value of the primary key for the table that was amended

7.3.2 ait_ctl_increment_log

This table is used to store the high level information about each increment. Its structure is described in Table 4

Table 4 - ait_ctl_increment_log table

Column	Usage
Increment_id	Primary key - identity column
Start_time	When the increment began
End_time	When the increment finished
Max_changes_log_id	What was the maximum primary key value in the amc_dm_changes_log table when the increment completed

7.3.3 ait_ctl_table_log

This table is used to store summary data on the operational copy tables (used as the source for the DataMart) at the beginning of each increment. The information in this table is predominantly used when transferring new operational rows to the DataMart. Its structure is described in Table 5. The data in this table should be maintained by the transfer procedures – please refer to section 7.3.10.

Table 5 - ait_ctl_table_log

Column	Usage
Table_log_id	Primary key - identity column
Increment_id	The increment that the row relates to
Table_name	The name of the operational copy table that the row relates to
Head_update_id	The maximum primary key value at the start of the increment
Tail_update_id	The minimum primary key value at the start of the increment - note that this will usually be 1 but that tables seeded by batch load may have a negative value.

7.3.4 ait_sum_new_tables

This table resides in the DataMart and is used to store the summary information about each table in the DataMart. Its structure is described in Table 6. It serves as the primary source for the summary display that is presented at the end of the DataMart creation (see Figure 12)

Table 6 – Ait_sum_new_tables structure

Column	Usage
Table_name	The name of the DataMart table
Entity_type	The type of table ('P' = Party, 'C' = Contract, 'RA' = Repeating attribute', 'T' = workflow, 'REF' = reference data, 'FIX' = other fixed tables i.e. engagements)
Ref_data_group_id	The reference data group that is used as the source for C, P, RA and REF tables
Ref_data_item_id	The reference data item that is used as the source for C, P and RA tables
Source_table_name	The name of the operational table that is primary source of the data
Primary_key_column	The name of the primary key column on the operational source table

7.3.5 ait_sum_new_columns

This table resides in the DataMart and is used to store the detailed information about each table in the DataMart. Its structure is described in Table 7. It serves as the secondary source for the summary display that is presented at the end of the DataMart creation (see Figure 12)

Table 7 – Ait_sum_new_columns

Column	Usage
Table_name	The name of the DataMart table
Column_name	The name of the column in the DataMart table
Data_type	The data type of the column
Is_Nullable	Is the column an optional (1) or compulsory field (null/0)?
Precision	Specifies the number of characters/digits that can be stored.
Scale	Specifies the maximum number of decimal digits that can be stored to the right of the decimal point (numeric data types only)
Cbe_definition_id	If the table is a P,C or RA entity type, what is the cbe_definition_id of the column
Ref_data_definition_id	If the table is a REF entity type, what is the ref_data_definition_id of the column

7.3.6 ait_sum_column_conflict

This table resides in the DataMart and is used to store information about any column name conflicts that were encountered and resolved. Its structure is described in Table 8.

Table 8 – Ait_sum_column_conflict

Column	Usage
Table_name	The name of the DataMart table where the conflict column was encountered

Column	Usage
Cbe_definition_id	The reference data group id that indicates the type of Portrait Configurable Business Entity (CBE) to which this table relates. If this table is for a reference data group rather than a CBE this column will be NULL.
Ref_data_definition_id	The reference data group id of the conflicting column, if it is a reference column, or NULL if it is a non-reference data type.
Hierarchical_prefix	Not currently used
Original_column	The name of the first column against which the conflict occurred
Conflict_column	The name of the conflicting column
Resolved_column	The name which was given to the conflicting column in order to resolve the conflict
Created_when	The date and time when this record was created
Updated_when	The date and time when this record was last updated

7.3.7 Triggers

Operational database triggers are required for each table in the operational database that you wish to track for updates and deletions. Therefore for the majority of tables two triggers will be required, one to record updates and one to record deletes. These triggers need to write key information into the `amc_dm_changes_log` table on the operational database. The log then serves as a key input into the incremental update process.

The triggers need to write three key pieces of data into this log table:-

1 Table_name : the table being amended

2 Operation : 'update' or 'delete'

3 Primary_key : The primary key value of the row being amended

The trigger syntax thus looks like this:-

```
CREATE TRIGGER t_imp_upd|del_<table name> ON <table_name>
FOR <update|delete> AS
INSERT INTO amc_dm_changes_log
( table_name, operation, primary_key )
SELECT '<table_name>', '<UPDATE|DELETE>', <table_primary_key>
FROM deleted
```

Where more than one table contains the data for a logical entity, the triggers on all tables write the same table name into the changes log, so that they can be treated uniformly. Thus the triggers on the Party tables `amc_pce_party_type_data` and `amc_pce_party` both write the table name '`amc_pce_party`' into the changes log.

7.3.8 SQL storage tables

These tables store the generated DDL and DML statements prior to their execution. They are seeded by the relevant stored procedures (see sections 7.3.9 and 7.3.10). The execution phase of the utility will then extract these statements from these tables and run them.

There are three tables

- `amc_dm_table_sql` – stores the 'Create table' DDL

- `amc_dm_data_sql` – stores DML to transfer data to the DataMart
- `amc_dm_post_data_sql` – stores 'CREATE CONSTRAINT' DDL

Each of the storage tables has the same structure as described in Table 9.

Table 9 - Structure of the storage tables

Column	Usage
sql_order	Primary key identity value - generated by SQL server when a row is inserted into the table.
Is_complete	Indicates whether the statement stored in sql_statement is a logically and syntactically complete SQL statement. Since there are limits on the maximum size of single VARCHAR variable and the maximum width of a single row in a table, it sometimes occurs that a statement that needs to be executed as part of the DataMart load process is too big to fit into a single row in one of the SQL storage tables. If this situation does arise, the statement should be split into two or more sections and inserted into consecutive rows in the relevant table. All rows except the last should have is_complete set to 0; the last row should have is_complete set to 1. When the statements are executed, the utility will join the statements back together in the order in which they were inserted and execute them as a single statement. Otherwise this column should always be set to 1.
Is_executed	Internal flag - do not use. This column has a default, so it is not necessary to specify a value for it.
Contains_conflict	Internal flag - do not use. This column has a default, so it is not necessary to specify a value for it.
Sql_statement	The actual SQL statement to be executed should be inserted into this column.
Start_time	The reported time at which the above sql statement started running
End_time	The reported time at which the above sql statement finished running

7.3.9 p_imp_dm_create_imp_tables

This procedure will need to exist on the operational database and will create the DDL to generate the desired implementation specific objects in the DataMart. This routine will only get called in a full creation of the DataMart as the tables will already exist when running an incremental update.

7.3.9.1 Parameters

Parameter	Data type	Meaning
@p_datamart	VARCHAR(50)	The name of the DataMart database.
@p_datamart_owner	VARCHAR(50)	The owner of the DataMart database.

7.3.9.2 Description

This procedure should generate SQL statements to create tables in the DataMart. For each implementation specific DataMart table, four kinds of statement need to be generated in the order shown.

1 "CREATE TABLE" DDL statements need to be generated and then stored in the ait_dm_table_sql

2 The "ALTER TABLE ADD CONSTRAINT" and "CREATE INDEX" DDL statements need to be generated and stored in ait_dm_post_data_sql.

3 A control statement, to insert table summary data into the summary table ait_sum_new_tables, needs to be generated and stored in ait_dm_table_sql.

4 A control statement to insert table structure data into the summary table ait_sum_new_columns, needs to be generated and stored in ait_dm_table_sql.

7.3.9.3 Example code fragment

Example code fragments, that illustrate each of these statements are shown in Fragment 1 – Fragment 4.

Note these important points

- When creating these statements, all table names should be prefixed with the name and owner of the DataMart (supplied as parameters)
- Square brackets must be used around the DataMart name (as shown below)
- All business data DataMart tables should be prefixed 'ait_dm_' in order to ensure that they are identified as business data tables by other sections of the utility.

Fragment 1 : An example 'create table' DDL statement

```
/* Note that the generated statement is stored in amc_dm_table_sql */

SET @v_sql = 'CREATE TABLE [' + @p_datamart + '].' + @p_datamart_owner
+ CHAR(13) + CHAR(10)+ '.ait_dm_my_table (my_primary_key INT NOT NULL, '
+ CHAR(13) + CHAR(10)+ 'my_data VARCHAR(255) NULL)'

INSERT INTO amc_dm_table_sql (is_complete, sql_statement) VALUES (1, @v_sql)
```

Fragment 2 : An example 'add constraint' DDL statement

```
/* Note that the generated statement is stored in amc_dm_post_data_sql. There may be
several of these statements, one per constraint/index on the DataMart table */

SET @v_sql = 'ALTER TABLE [' + @p_datamart + '].' + @p_datamart_owner
+ CHAR(13) + CHAR(10)+ '.ait_dm_my_table ADD CONSTRAINT pk_ait_dm_my_table '
+ CHAR(13) + CHAR(10)+ 'PRIMARY_KEY (my_primary_key)'

INSERT INTO amc_dm_post_data_sql (is_complete, sql_statement) VALUES (1, @v_sql)
```

Fragment 3 : An example 'ait_sum_new_tables' control statement

```
/* Note that the generated statement is stored in amc_dm_table_sql */

SET @v_sql = 'INSERT INTO ait_sum_new_tables (table_name, entity_type,
     source_table_name, primary_key_column)'
+ CHAR(13) + CHAR(10) + 'VALUES ("ait_dm_my_table","FIX","<operational source
     table>","operational_primary_key"'

INSERT INTO amc_dm_table_sql (is_complete, sql_statement) VALUES (1, @v_sql)
```

Fragment 4 :An example 'ait_sum_new_columns' control statement

```
/* Note that the generated statement is stored in amc_dm_table_sql
There may be several of these statements, one per column in the DataMart table */

SET @v_sql = 'INSERT INTO ait_sum_new_columns (table_name, column_name,
     data_type, is_nullable)'
+ CHAR(13) + CHAR(10) + 'VALUES ("ait_dm_my_table",'"my_primary_key",'Integer",0)'

INSERT INTO amc_dm_table_sql (is_complete, sql_statement) VALUES (1, @v_sql)
```

7.3.10 p_imp_dm_transfer_imp_data

This procedure will need to exist on the operational database and will create the DML statements to transfer the data in the desired implementation specific tables into the DataMart. This procedure will be called by both a full creation and an incremental update of the DataMart.

7.3.10.1 Parameters

Parameter	Data type	Meaning
@p_source_database	VARCHAR(50)	The name of the Portrait database.
@p_datamart	VARCHAR(50)	The name of the DataMart database.
@p_source_owner	VARCHAR(50)	The owner of the Portrait database.
@p_datamart_owner	VARCHAR(50)	The owner of the DataMart database.
@p_last_increment_id	INTEGER	The last increment number.

7.3.10.2 Description

This procedure should generate SQL statements to populate the implementation-specific tables in the DataMart. The generated statements should all be placed in the amc_dm_data_sql table.

For each implementation specific DataMart table, three kinds of statement need to be generated in the order shown.

1. A delete DML statement to remove rows that have been deleted or updated in the operational system – stored in amc_dm_data_sql.
2. An insert DML statement to insert rows that have changed or been added to the operational system – stored in amc_dm_data_sql.
3. An control statement that maintains the information in ait_ctl_table_log – stored in amc_dm_data_sql.

7.3.10.3 Example code fragments

Example code fragments, that illustrate each of these statements are shown in Fragment 5 – Fragment 7.

Note these important points

- All references to tables in the DataMart should be prefixed with the name and owner of the DataMart (supplied as parameters)
- Square brackets must be used around the DataMart name
- All references to tables in the operational copy should be prefixed with the supplied source database name and owner (supplied as parameters)

Fragment 5 : An example delete DML statement

```
/* Need to delete all rows that have been update or deleted (reference the
amc_dm_changes_log table) in this increment (reference the ait_ctl_increment_log table) */

DELETE FROM [<@p_datamart >].<@p_datamart_owner>.[<DataMart table>]
WHERE <primary key> IN
( SELECT primary_key
FROM [<@p_datamart >].<@p_datamart_owner>.amc_dm_changes_log
WHERE table_name = '<operational copy table name>
AND changes_log_id >
(SELECT TOP 1 ISNULL(max_changes_log_id,0)
FROM[<@p_datamart>].<@p_datamart_owner>.ait_ctl_increment_log
WHERE increment_id =<@p_last_increment_id> ) )
```

Fragment 6 : An example insert DML statement

```
/* Need to insert all rows that are new (reference the ait_ctl_table_log) and all rows
that have been amended (reference the amc_dm_changes_log table), in this increment
(reference the ait_ctl_increment_log table) */

INSERT INTO [<@p_datamart>].<@p_datamart_owner> (<column list>)
SELECT <standard select statement>
FROM <table list>
LEFT OUTER JOIN [<@p_datamart>].<@p_datamart_owner>.amc_dm_changes_log cl
ON cl.table_name  = 'operational copy table name'
AND eap.<operational table primary key> = cl.primary_key
AND cl.operation = 'UPDATE'
AND cl.changes_log_id > <ait_ctl_increment_log.max_changes_log_id>
WHERE ( <operational copy table name> > <ait_ctl_table_log.head_update_id>
OR <operational copy table name>.<primary key> < <ait_ctl_table_log.tail_update_id>
OR cl.primary_key IS NOT NULL )
```

Fragment 7 : An example 'ait_ctl_table_log' control statement

```
/* The ait_ctl_table_log needs to be updated with the minimum and maximum primary key
values for the source operational copy tables – this way, the next time an increment is
run, the insert routines have the last increment primary key values and thus know the
primary key values which the increment needs to insert from. */

INSERT INTO [<@p_datamart>].<@p_datamart_owner>.ait_ctl_table_log (table_name,
head_update_id, tail_update_id, increment_id)
SELECT '<operational_copy_table_name>', max(<primary key>), min(<primary key>),
(SELECT max(increment_id)
FROM [<@p_datamart>].<@p_datamart_owner>.ait_ctl_increment_log)
FROM [<@p_source_database>].<@p_source_owner>.<operational_copy_table_name>
```

7.3.11 amc_dm_opt_cbe_options

This table is used to store the options as to which Portrait Configurable Business Entities (CBEs) should be exported to the DataMart. This table is stored in the operational database, and a copy written to the DataMart when it is created. When the DataMart is subsequently updated, the same options are therefore used as when it was created. Its structure is described in Table 10 although it also includes the standard audit columns.

Table 10 - amc_dm_changes_log table

Column	Usage
cbe_type_rdg	The reference data group id of this CBE. This defines which type of entity it is, for example a Party, a Contract, etc
cbe_type_rdi	The reference data item id of this CBE. This defines which one of a particular type of entity this is, for example the Consumer Party. If this column is zero, the record represents the top level tree nodes in the DataMart rebuild settings dialog, and the column export_cbe contains 1 if any of its child nodes are selected.
cbe_type_name	The name of this entity as displayed in the DataMart rebuild settings dialog
export_cbe	1 if this CBE is to be exported, 0 otherwise

7.3.12 amc_dm_opt_gen_options

This table is used to store the general options used to create the DataMart, and has a single row. It is stored in the operational database, and a copy written to the DataMart when it is created. When the DataMart is subsequently updated, the same options are therefore used as when it was created. Its structure is described in Table 11.

Table 11 – amc_dm_changes_log table

Column	Usage
option_id	The Primary Key
use_display_names	1 if display names are used in creating the DataMart table and column names, 0 if system names are used.
encrypt_ids	1 if primary keys are to be encrypted in the DataMart tables, 0 if they are to remain unencrypted.
option_xml	For future expansion, to store further options

Appendix A Troubleshooting

Most DataMart issues are likely to be either functional or performance related. This appendix offers hints and tips on how to determine and resolve these problems.

A.1 Functional problems

The DataMart is quite simple in concept and most functional problems can be tracked down to one of the three key phases.

1. Setup and connectivity – these errors are usually to do with incorrect inputs i.e. the usernames or database names are wrong.
2. SQL statement generation – these errors normally indicate a problem running a stored procedure on the DataMart database. The extract should be repeated and the DataMart database traced using SQL profiler.
3. SQL statement execution – these errors normally indicate a problem executing one of the generated statements on the DataMart. The 'trouble statement' should be located by running a utility script (*amc_utl_datamart_functional_diagnosis.sql*) on the DataMart database.

A.1.1 Common errors

Common error messages and troubleshooting steps are shown in Table 12. The same errors will be reported regardless of whether batch or interactive mode is used.

Table 12 – Troubleshooting common errors

Error Messages	How to diagnose/correct
Failed to connect to the Portrait database. Please check the operational database settings. Failed to connect to the DataMart database. Please check the DataMart database settings.	Are the user accounts that you are using to connect to the database and the DataMart members of the SQL Server System Administrator server role? Do both the operational copy database and the DataMart database exist? (check the spellings). Are they available?
One or more of the Operational database's DataMart triggers has been disabled. This will potentially mean that the resulting DataMart will not contain accurate data. Are you sure that you want to ignore this and continue?	You should ensure that you understand why one of the DataMart triggers has been disabled before continuing. They can all be re-enabled by running the utility procedure **p_amc_utl_disable_dm_changes_log** and setting its *@p_disable_it* parameter to 0. However, this may be a symptom of the DataMart functionality not having been enabled during the operational database installation, or that certain triggers have manually been disabled. Contact Portrait Support for more information.
It has not been possible to save your DataMart settings back to the operational database for the following reason: … However, this only means they will not be persisted for when you next create a DataMart - you may still continue with the DataMart creation process.	This error message displays any database error that is encountered while saving the DataMart settings to the operational database - for example if the operational database you are using is read-only. It tells you that this is not a serious problem, but that the options used in creating the DataMart will not be persisted as the defaults for the next time you create one. It does not stop you continuing with creation of the DataMart, and the options you have chosen will be used when doing so.
The attempt to generate the SQL to create the DataMart structure has failed	Repeat the DataMart extract tracing the DataMart database with a SQL profile Attempt to obtain further error information by manually running the 'trouble statement' in Query Analyser against the DataMart database. This is likely to be the call to the **p_amc_dm_create_tables** procedure.
The attempt to generate the SQL to populate the DataMart has failed	Repeat the DataMart extract tracing the DataMart database with a SQL profile Attempt to obtain further error information by manually running the 'trouble statement' in Query Analyser against the DataMart database. This is likely to be the call to the **p_amc_dm_transfer_data** procedure.
The attempt to execute the SQL to create the DataMart has failed The attempt to execute the SQL to populate the DataMart has failed	Run the amc_utl_datamart_functional_diagnosis.sql script against the DataMart database. The first record returned is likely to be the 'trouble statement'. Attempt to obtain further error information by manually running the 'trouble statement' in Query Analyser against the DataMart database.

A.1.2 SQL Profiler

SQL Profiler is a SQL Server tool that traces the events that occur on a specified database server. The profile can be configured to limit the type of events, databases and users that are traced.

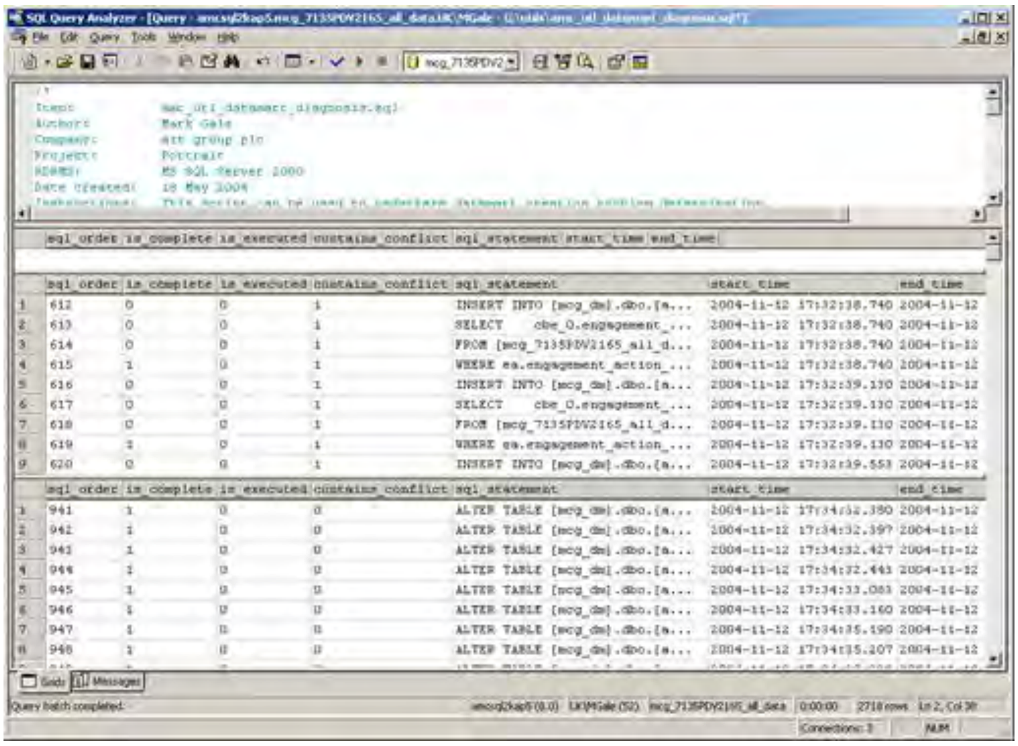
When diagnosing the DataMart, a limited set of events are traced and normally only the DataMart database need be traced. Guidance on how to undertake a diagnosing SQL profile is provided on the Foundation install media under Software\Tools\Database\Stored_Procedure_Analyser\ Datamart_extract_diagnosis.txt.

A.1.3 amc_utl_datamart_functional_diagnosis

This script can be found on the CD-ROM in Software\Tools\Database\useful_scripts.

The script searches for generated SQL statements that have not been executed and it should be run against the DataMart database. It returns three result grids one for table creation, data transfer and index/foreign key creation respectively. The first statement returned is normally the 'trouble statement'.

Figure 14 – amc_utl_datamart_functional_diagnosis output



```
/*
Script:          amc_utl_datamart_diagnosis.sql
Author:          Mark Gale
Company:         att group plc
Project:         Portrait
RDBMS:           MS SQL Server 2000
Date created:    18 May 2004
[illegible]      This Script can be used to [illegible]
```

```
  sql_order is_complete is_executed contains_conflict sql_statement start_time end_time
```

```
  sql_order is_complete is_executed contains_conflict sql_statement                   start_time              end_time
1 612       0           0           1                 INSERT INTO [mcg_dm].dbo.[a...  2004-11-12 17:32:38.740 2004-11-12
2 613       0           0           1                 SELECT    cbe_0.engagement_...  2004-11-12 17:32:38.740 2004-11-12
3 614       0           0           1                 FROM [mcg_7135PDV2165_all_d...  2004-11-12 17:32:38.740 2004-11-12
4 615       1           0           1                 WHERE ea.engagement_action_...  2004-11-12 17:32:38.740 2004-11-12
5 616       0           0           1                 INSERT INTO [mcg_dm].dbo.[a...  2004-11-12 17:32:39.130 2004-11-12
6 617       0           0           1                 SELECT    cbe_0.engagement_...  2004-11-12 17:32:39.130 2004-11-12
7 618       0           0           1                 FROM [mcg_7135PDV2165_all_d...  2004-11-12 17:32:39.130 2004-11-12
8 619       1           0           1                 WHERE ea.engagement_action_...  2004-11-12 17:32:39.130 2004-11-12
9 620       0           0           1                 INSERT INTO [mcg_dm].dbo.[a...  2004-11-12 17:32:39.553 2004-11-12
```

```
  sql_order is_complete is_executed contains_conflict sql_statement                   start_time              end_time
1 941       1           0           0                 ALTER TABLE [mcg_dm].dbo.[a...  2004-11-12 17:34:32.380 2004-11-12
2 942       1           0           0                 ALTER TABLE [mcg_dm].dbo.[a...  2004-11-12 17:34:32.397 2004-11-12
3 943       1           0           0                 ALTER TABLE [mcg_dm].dbo.[a...  2004-11-12 17:34:32.427 2004-11-12
4 944       1           0           0                 ALTER TABLE [mcg_dm].dbo.[a...  2004-11-12 17:34:32.443 2004-11-12
5 945       1           0           0                 ALTER TABLE [mcg_dm].dbo.[a...  2004-11-12 17:34:33.083 2004-11-12
6 946       1           0           0                 ALTER TABLE [mcg_dm].dbo.[a...  2004-11-12 17:34:33.160 2004-11-12
7 947       1           0           0                 ALTER TABLE [mcg_dm].dbo.[a...  2004-11-12 17:34:35.190 2004-11-12
8 948       1           0           0                 ALTER TABLE [mcg_dm].dbo.[a...  2004-11-12 17:34:35.207 2004-11-12
```

In this example (Figure 14) the first result grid is empty – this suggests that the DataMart tables were created successfully. The first record returned is in the second grid (sql_order 612) suggesting that the data transfer step has broken down mid-way through. It may be possible to obtain error further information by pasting this 'trouble statement' into another window and executing it against the DataMart. Please note :-

- To obtain the full statement you must set the **maximum character per column** Query Analyser settings to 8192 (tools→options→results)
- The full statement may exist over more than one line in the result set. In Figure 14, the 'trouble statement' starts at sql_order 612 but ends with sql_order 615. This is indicated by the is_complete =1 entry in sql_order 615. Consequently all 4 lines (612→615) will need to be pasted to a new window and run against the DataMart.

A.2 Performance problems

The DataMart extract retrieves a vast amount of data from the operational copy database and inserts it into a new structure in the DataMart database.

This is a potentially very large task and it is not possible to predict how long this will take – hardware specification, network bandwidth and operational database size are all variable factors that influence extract duration. As a consequence it is difficult to determine if a particular extract is performing badly or not.

First of all, make sure that in the DataMart settings dialog (see section 6.1.3 for further details) you have only selected those entities for export that you **actually** require. In particular, the export of Task history, Composite data object types and Task types is especially time-consuming, and these should be excluded if they are not required.

It is possible to study the durations of the steps within the extract and determine which are the most costly – the most costly ones can then be analysed in search of possible performance enhancements. This can be achieved by using a combination of bespoke Portrait Foundation scripts and SQL Server's Query Analyser.

To do this (requires moderate SQL skills):

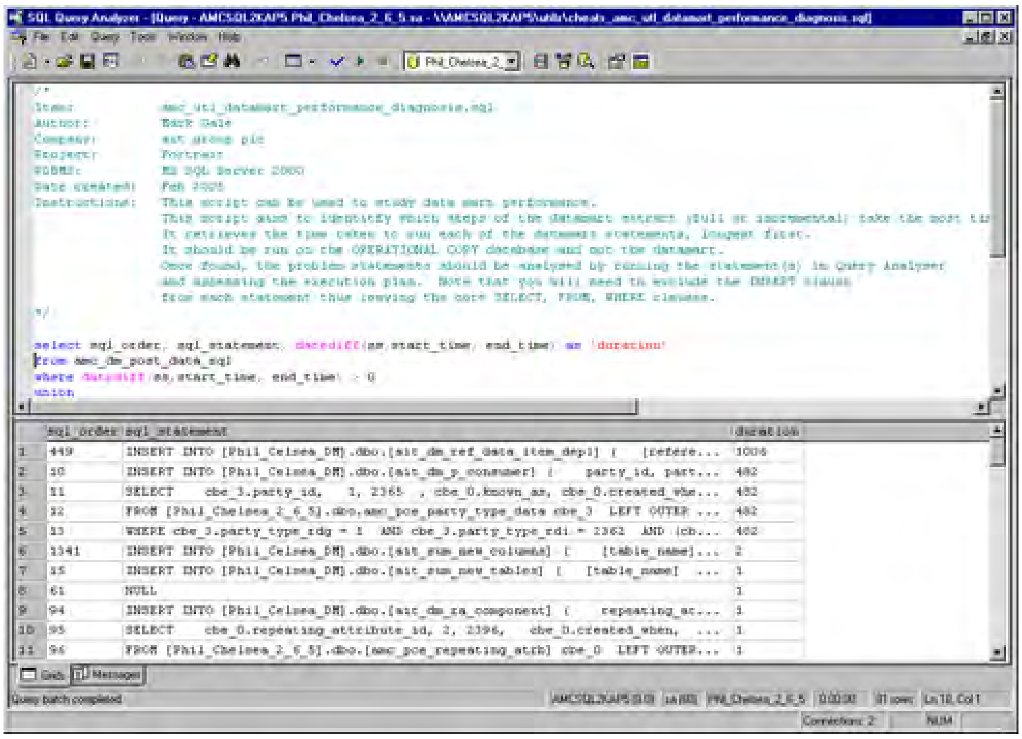
1. Open Query Analyser against the DataMart database and run the utility script *amc_utl_dm_performance_diagnosis.sql* (this script can be found on the CD-ROM in Software\Tools\Database\useful_scripts). This script searches the extract's SQL statements and returns the statements in order of descending duration.
2. Copy the expensive statements (those that have higher durations) into a fresh Query Analyser window against the DataMart database Note that most of the statements will start with an INSERT clause – this can be discarded as it's only the core SELECT, FROM, WHERE statement that we wish to study.
3. Ensure that the Query Analyser has **Show Execution Plan** selected (CTRL+M) and execute the statements.
4. Analyse the execution plans via the execution plan pane and search for the most costly steps in each statement – statements that use table scan actions are prime candidates for performance tuning.
5. If you think that the execution plans indicate that potential performance gains are possible (i.e. they show table scans), please send the statement(s) in question along with its execution plan to portrait .support@aitgroup.com. The easiest way is to despatch this information is to turn **Show Execution Plan** off (CTRL+M), SET SHOWPLAN_TEXT on (in SSMS choose Tools | Options | Query Execution | SQL Server | Advanced), re-run the query and copy the resulting output into a text file. Please note that improving performance will often require a database patch to add additional indexes to the operational database – under NO circumstances should a project attempt to indexes them themselves.

A.2.1 amc_utl_datamart_performance_diagnosis

This script can be found on the CD-ROM in Software\Tools\Database\useful_scripts.

This script searches the extract's SQL statements and returns the statements in order of descending duration – it should be run against the DataMart database.

Figure 15 – amc_utl_performance_diagnosis output



SQL Query Analyzer - [Query - AMCSQL2KAPS Phil_Chelsea_2_6_5 sa - \\AMCSQL2KAPS\utils\cheats_amc_utl_datamart_performance_diagnosis.sql]

```
/*
Name:           amc_utl_datamart_performance_diagnosis.sql
Author:         Mark Dale
Company:        ait group plc
Project:        Portrait
RDBMS:          MS SQL Server 2000
Date created:   Feb 2005
Instructions:   This script can be used to study data mart performance.
                This script aims to identify which steps of the datamart extract (full or incremental) take the most ti
                It retrieves the time taken to run each of the datamart statements, longest first.
                It should be run on the OPERATIONAL COPY database and not the datamart.
                Once found, the problem statements should be analysed by running the statement(s) in Query Analyser
                and assessing the execution plan.  Note that you will need to exclude the INSERT clause
                from each statement thus leaving the core SELECT, FROM, WHERE clauses.
*/

select sql_order, sql_statement, datediff(ss,start_time, end_time) as 'duration'
from amc_dm_post_data_sql
where datediff(ss,start_time, end_time) > 0
union
```

```
    sql_order sql_statement                                                                duration
1   449       INSERT INTO [Phil_Celsea_DM].dbo.[ait_dm_ref_data_item_dep1] (   [refere...  1006
2   10        INSERT INTO [Phil_Celsea_DM].dbo.[ait_dm_p_consumer] (    party_id, part...  482
3   11        SELECT    cbe_3.party_id,    1, 2365  , cbe_0.known_as, cbe_0.created_whe...  482
4   12        FROM [Phil_Chelsea_2_6_5].dbo.amc_poe_party_type_data cbe_3  LEFT OUTER ...  482
5   13        WHERE cbe_3.party_type_rdg = 1  AND cbe_3.party_type_rdi = 2362  AND (cb...  482
6   1341      INSERT INTO [Phil_Celsea_DM].dbo.[ait_sum_new_columns] (    [table_name]...  2
7   15        INSERT INTO [Phil_Celsea_DM].dbo.[ait_sum_new_tables] (   [table_name]  ...  1
8   61        NULL                                                                         1
9   94        INSERT INTO [Phil_Celsea_DM].dbo.[ait_dm_ra_component] (    repeating_at...  1
10  95        SELECT    cbe_0.repeating_attribute_id, 2, 2396,   cbe_0.created_when,  ...  1
11  96        FROM [Phil_Chelsea_2_6_5].dbo.[amc_poe_repeating_atrb] cbe_0  LEFT OUTER...  1
```

Query batch completed | AMCSQL2KAPS (8.0) | sa (60) | Phil_Chelsea_2_6_5 | 0:00:00 | 87 rows | Ln 18, Col 1 | Connections: 2 | NUM

In this example (Figure 15) two queries are clearly the worst performing and are candidates for performance tuning. Please note :-

- To obtain the full statement you must set the **maximum character per column** Query Analyser settings to 8192 (tools→options→results)
- The full statement may exist over more than one line in the result set. In Figure 15, the second statement is actually spread over four rows (sql_order 10 – 13). Sql_order 10 (the INSERT clause) can be discarded and consequently 3 lines (11-13) will need to be pasted to a new Query Analyser window and run against the DataMart database with **show execution plan** enabled.

A.3 If symptoms persist….

Should the suggestions in this appendix not help resolve the matter, please contact Portrait Support (Portrait.Support@portraitsoftware.com).

Please ensure that you pass on the following information

1. Confirmation that you have examined the common errors in Table 12 and eliminated them as possible causes of your error.
2. Which release of Portrait Foundation you are using.
3. Whether you are using
   - Batch or interactive mode
   - SQL Server or Windows authentication
   - Full creation (complete rebuild) or an incremental update.
4. The error message you are presented with
5. Output from SQL Profiler, amc_utl_datamart_functional_diagnosis.sql and amc_utl_dm_performance_diagnosis.sql as appropriate
6. What has changed since the extract last worked

Any other information that you consider may be useful for swift problem determination.