Portrait
Foundation

Database Administrators Guide

Edition 19.0

04 March 2014

Pitney Bowes
Software

Portrait Foundation
Database Administrators 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

Portrait Software
The Smith Centre
The Fairmile
Henley-on-Thames
Oxfordshire, RG9 6AB, UK

Email: support@portraitsoftware.com
Tel: +44 (0)1491 416778
Fax: +44 (0)1491 416601

America

Portrait Software
125 Summer Street
16th Floor
Boston, MA 02110
USA

Email: support@portraitsoftware.com
Tel: +1 617 457 5200
Fax: +1 617 457 5299

Norway

Portrait Software
Portrait Million Handshakes AS
Maridalsveien. 87
0461 Oslo
Norway

Email: support@portraitsoftware.com
Tel: +47 22 38 91 00
Fax: +47 23 40 94 99

About this document

Purpose of document

This document is intended to act as a source of reference for those who will be administering Portrait Foundation databases in development, test and production environments.

It contains general advice on administering SQL Server databases in addition to detail that is specific only to Portrait Foundation.

Intended audience

This document is primarily aimed at database and system administrators responsible for the running of Portrait Foundation CRM databases.

It is also recommended reading for system architects and others involved in the design of Portrait Foundation systems.

Related documents

Database Setup Guide

Software release

Portrait Foundation 5.0 or later.

Contents

1 Database Administration

This section discusses some of the key concepts of the Portrait Foundation database – it is anticipated that they will be of most interest to those whose administer production environments. When working in a development and/or functionality testing environment a number of the points raised will not be relevant. It is useful, however, to have an appreciation of these issues since they may affect decisions made during the implementation process.

1.1 Software configuration

1.1.1 Requirements

The Portrait Foundation database server requires the following software. Please note that this listing only discusses supported platforms at a high level. The latest detailed platform support definition, including versioning and service packs, can be found in the *Release Notes.*

The database server requires:

- The supported Windows operating system; there is no support for non-Windows operating systems. See the 'Portrait Installation Guide' document for details.
- The supported version of SQL Server; see the 'Portrait Installation Guide' document for details.
- The database server should have a full install of SQL Server, including the SQL Server Management Studio for DBA administration.
- SQL Server should be operating in mixed authentication mode that is, both Windows and SQL Server authenticated users are permitted.

1.1.2 OS optimisation

Please note that these settings are unlikely to yield any significant benefit in development and test environments and thus may be ignored in these circumstances.

Portrait recommends that production servers use certain Microsoft Windows configuration settings to enable SQL Server to work most effectively. Note that failing to undertake these settings will not invalidate your Portrait Foundation environment and that they are simply recommendations to optimise your environment's performance.

Services

Select the Services icon and review the services that are automatically started:

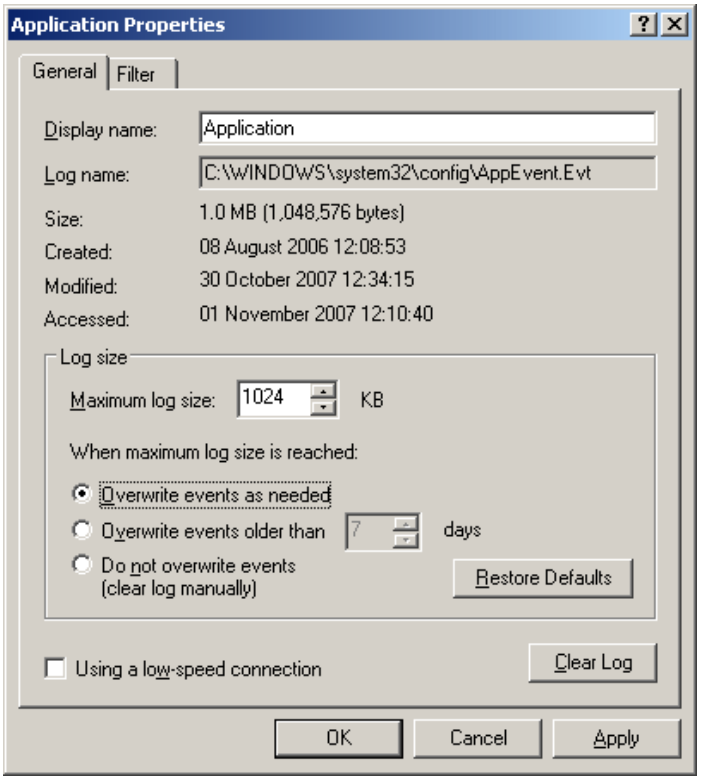
1. Ensure the following are automatically started and are running: Schedule, Server, Workstation, and Event Log.
2. Stop the Messenger Service and set its Start-up to Manual (as unanswered messages prevent a remote shutdown of the server).
3. Stop and Disable the Computer Browser Service (it has a performance impact on SQL Server).
4. Ensure that SQL Server and Server agent services are on automated startup.
5. Review the need for any other services as they may take up unnecessary resource. Note it is easier to remove unnecessary services early on in the life cycle, they can always be enabled if found to be required later.

Event Logs

Each of the three Microsoft Windows Event Logs (system, application and security) should be set to 'overwrite events as needed' thus minimising the risk of

downtime due to full logs. This can be achieved by invoking the **Event Viewer** (My Computer, Manage, System tools, Event Viewer) and right clicking on each of the three logs. Select the 'overwrite events as needed' option as illustrated in Figure 1.

Figure 1 – Event Log properties

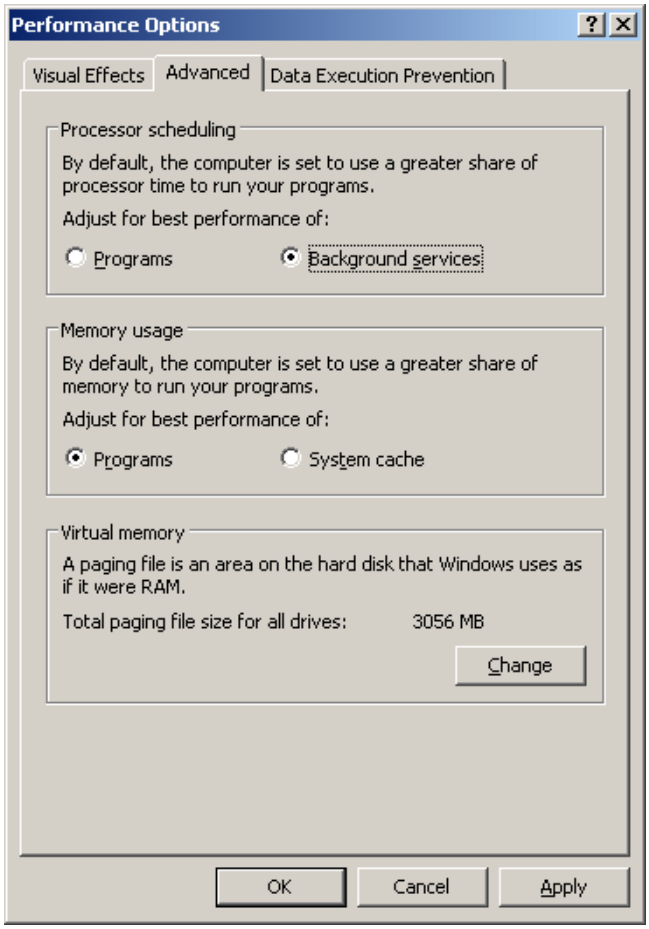


System properties

The server should be configured to optimise performance for SQL Server.

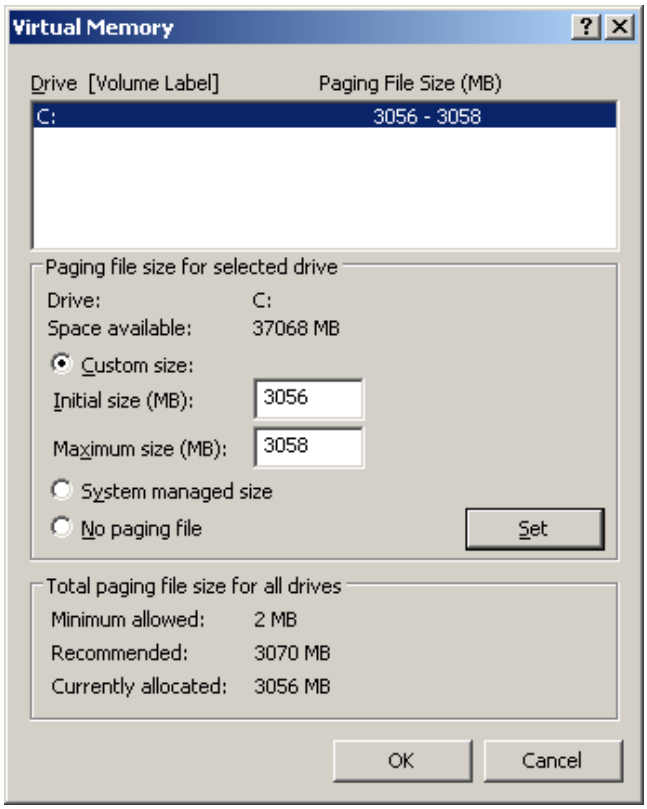
1 Right click on the **My Computer** icon and select properties. Choose the advanced tab and click Performance options. Select 'optimise performance for Background services' as illustrated in Figure 2.

Figure 2 – Optimize performance for Background services



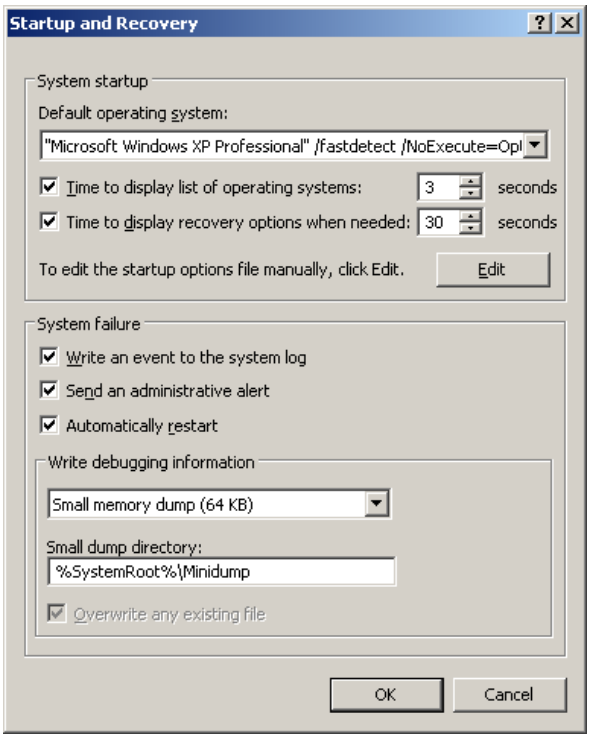
2 Click the change button to review the Virtual Memory options (illustrated in Figure 3) If necessary set up additional paging files to accommodate 1.5 times the amount of RAM installed in the computer, as recommended by Microsoft. Note that the paging file should be on a different disk to the data and log files – typically it would be best located on the same disk as the OS binaries. This change should be done in consultation with the Windows administrator.

Figure 3 – Paging file sizing



3 Select Advanced tab and click Settings in the Startup and Recovery section. Change 'time to display list of operating systems' time to 3 seconds. This will shorten reboot time, whilst still allowing sufficient time to intervene. For 'system failure' check all three check boxes and 'select Small memory dump' for the amount of debugging information. Refer to Figure 4.

Figure 4 – Startup and Recovery settings



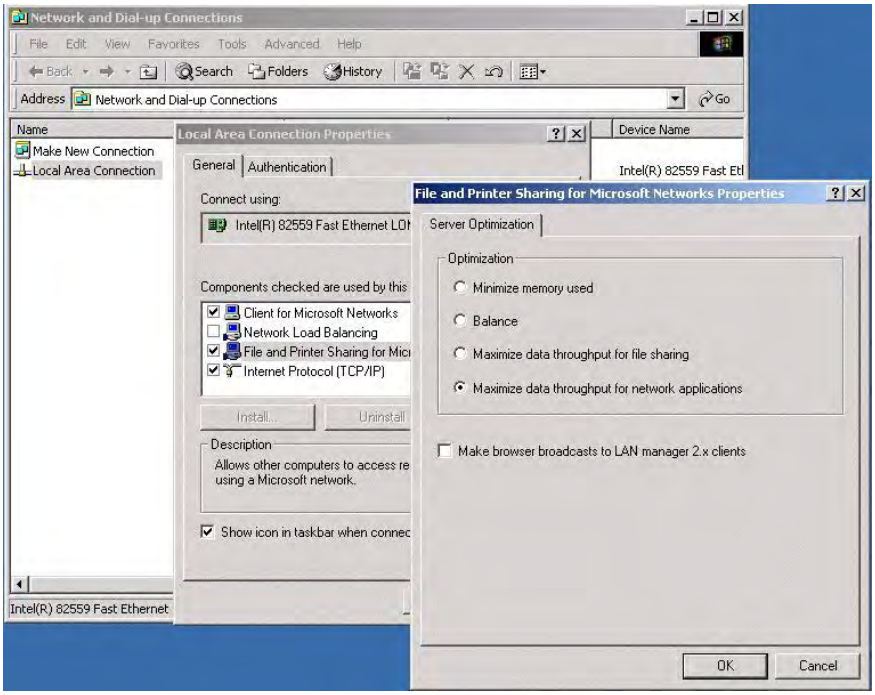
Network properties

The network properties should be optimised for Network applications thus ensuring that SQL Server does data caching and Windows does not need to do its own.

1 Select **My Network Places**, select **Local Area Connection**, and then select **File and Print Services for Microsoft Networks**.

2 Right click and select **Properties**, and then **Maximise data throughput for Network Applications**.

Figure 5 – Network properties



1.2 File groups

The Portrait Foundation operational database contains the following file groups to try and separate the different types of data stored by the system.

There are 2 types of group

- DATA - contains the table and stored procedure objects
- INDEX - contains the indexes and foreign key constraints

Name	Description
BATCH	Contains Batch Load related objects
BUSINESS	Contains instances of configured business entities like Parties, Engagements, Repeating Attributes, Milestones, Contracts and Product Definitions
CAMPAIGNS	Contains decisions engine related objects
CONFIG	Contains deployed configuration and data including Smart Lookups and Reference Data
IMP	Project Implementation specific objects – not used by Portrait Foundation
OPERATIONS	Contains Session data, Model Instance data, Task data, Blob data, Client Events data and Composite Data Objects

1.3 Sizing

An accurate estimate of the database size has several advantages in that it:

- Enables the accurate planning of server disk space.
- Permits forecasting on system resources (CPU and memory) which is necessary to facilitate good system performance.
- Allows efficient scheduling of backup routines.

There are several key inputs into the sizing process most of which are influenced by business volume projections. These inputs include customer volumes and types, contract volumes and product types and, most significantly, the number and complexity of business process models.

It's difficult to know how a Project Implementation's database is going to grow as the configured objects and processes are always different. The tables that contain instance data are always going to grow, but that does depend again on the usage of the system. So over time you should see the BUSINESSDATA & OPERATIONSDATA groups to consistently grow.

The following SQL can be used to inspect a Portrait Foundation database and can be found on the Release CD under \Software\Tools\Database\useful_scripts

- amc_utl_data_sizes.sql
- amc_utl_file_group_size.sql
- amc_utl_table_space_usage.sql

They output size information about different types of data in the database so you can see what's taking up space and can predict the effect of increasing the number Portrait Foundation objects in the system.

1.4 File placement

The treatment of data and transaction log files, SQL Server binaries, **tempdb** and so on, can dramatically improve or degrade the overall performance of a SQL database server. While individual implementations will vary, this section provides a suggested disk and installation configuration for a database server.

The following table lists the distinct physical disk requirements along with their purpose. Suggested capacities of the disks are not included since these will depend on the nature of the implementation.

Table 1 – Physical disk requirements

Physical disk	RAID level	Purpose
0	RAID 1	Operating system and page file
1	RAID 0	SQL Server binaries and **tempdb**
2	RAID 1	Database transaction log files
3	RAID 1+0	Data files (tables and indexes)

Follow these principles when deciding on your own configuration:

- **Reduce disk contention** – for example, keep the data files and the transaction log files on separate physical disks.
- **Consider fault tolerance** – usually a trade off between the value of your data, time constraints imposed for restoring data and budget constraints.
- **Extensibility** – be aware of future requirements to increase the capacity of the server.

Another issue worth considering is the initial sizing of the database files. It is advisable to create the data and transaction log files to be the maximum possible size. Pre-sizing of these files removes the need for SQL Server to grow the files

automatically as they fill up and thereby improves performance. This approach will require that file usage be monitored so that warning can be given when the server is running out of space.

1.5 Backup and recovery

A viable backup and recovery strategy is vital for a production database. The exact nature of the strategy will depend on factors such as service level agreements (for example, maximum permissible downtime), the nature of database server hardware, availability of peripheral devices such as tape drives, the volume of the data to be backed up/recovered, use of third-party software and so on.

It is generally necessary to be able to recover to the point in time where the database failure occurred. For this reason it is recommended that SQL Server's full recovery model is selected as the standard. On occasion, for example during an initial data load, it may be advisable to switch to another recovery model such as bulk-logged. This approach will assist in conserving transaction log space and improving performance.

The SQL Server Books Online documentation contains more detailed information to assist in designing and implementing a backup and recovery strategy that will suit the individual requirements of a particular implementation. As a starting point it is recommended that a complete backup of the Portrait Foundation database is taken once a day. Although backups can be taken while the database is operational it is recommended that complete backups be scheduled during quiet periods so as to minimise any impact on database performance.

The following example (taken from SQL Server Books Online) describes a more complex strategy. This technique makes use of complete, differential and transaction log backups in order to reduce the amount of time required to restore the database. Portrait recommends that this strategy be adopted in Portrait Foundation implementations.

A mission-critical database system requires that a complete database backup is created each night at midnight, a differential database backup is created on the hour, Monday through Saturday, and transaction log backups are created every 10 minutes throughout the day. If the database needs to be restored to its state at 5:19 A.M. on Wednesday:

1. Restore the database backup created on Tuesday night.
2. Restore the differential database backup created at 5:00 A.M. on Wednesday.
3. Apply the transaction log backup created at 5:10 A.M. on Wednesday.
4. Apply the transaction log backup created at 5:20 A.M. on Wednesday, specifying that the recovery process only applies transactions that occurred before 5:19 A.M.

Alternatively, if the database needs to be restored to its state at 3:04 A.M. on Thursday, but the differential database backup created at 3:00 A.M. on Thursday is unavailable for some reason:

1. Restore the database backup created on Wednesday night.
2. Restore the differential database backup created at 2:00 A.M. on Thursday.
3. Apply all the transaction log backups created from 2:10 A.M. to 3:00 A.M. on Thursday.
4. Apply the transaction log backup created at 3:10 A.M. on Thursday, specifying that the recovery process only applies transactions that occurred before 3:04 A.M.

1.6 Collation issues

When setting up a Portrait Foundation database it is necessary to pay attention to the collation defined for the server and the databases within it.

The Portrait Foundation database contains a limited number of stored procedures that use temporary tables. When a temporary table is created in SQL Server it physically resides within **tempdb**. If **tempdb** has a different collation to the Portrait Foundation database, any SQL that attempts to join the temporary table to user tables in the Portrait Foundation database using character columns (**VARCHAR**, **NVARCHAR**, **CHAR**, **NCHAR**) will fail. For this reason it is vital that the collation name of the Portrait Foundation database matches the Server collation.

1.7 Monitoring

Monitoring the database and the server upon which it resides is essential in order to maintain consistent levels of performance and to obtain early warnings of impending problems.

Windows System Monitor can be used to monitor system resources; this tool allows administrators to view resource usage data in a graphical format. The SQL Server Books Online documentation contains more detailed information on this tool and indeed the monitoring of specific system resources.

As a starting point, consideration of the following resources is recommended:

- Disk activity (I/O and paging)
- CPU usage
- Memory usage
- User connections
- Locking

1.8 Maintenance plans

It will be necessary to run maintenance routines periodically, especially on production databases. SQL Server provides a wizard to assist in designing and scheduling maintenance plans.

A maintenance plan can consist of a number of distinct tasks. How frequently the maintenance tasks need to be run will depend on database usage, size and performance requirements. It is also likely that requirements will change over time. Close monitoring of the Portrait Foundation database will provide information that can be used to tune how frequently the tasks need to be executed.

As a starting point it is suggested that the following tasks be scheduled to run at least once per week:

- Reorganising the data on the data and index pages by rebuilding indexes with a new fill factor. This ensures that database pages contain an equally distributed amount of data and free space, which allows future growth to be faster.
- Updating index statistics to ensure the query optimizer has up-to-date information about the distribution of data values in the tables. This allows the query optimizer to make better judgements about the best way to access data because it has more information about the data stored in the database. Although index statistics are automatically updated by SQL Server on a periodic basis, this option can force the statistics to be updated immediately.
- Performing internal consistency checks of the data and data pages within the database to ensure that a system or software problem has not damaged data.

Again, the SQL Server Books Online documentation contains more detailed information to assist with designing and implementing maintenance plans.

1.9 Database resilience

The Portrait Foundation database is the focal point in an implementation of the product. It provides the central source of customer, account, product and system data that can be viewed by the variety of channels that have been configured. While it is desirable to have data stored only once in a single location this can bring its own problems since the database then becomes a potential single point of failure.

There are a number of methods that can be used to ensure that the Portrait Foundation database server can withstand common risks of failure.

- Fault tolerant disk arrays have already been mentioned as a method of ensuring that data is secure.
- Fail-over clustering makes use of hardware redundancy. Two servers are configured as one node sharing a common disk array(s). Although this option provides a robust solution it can prove to be expensive.
- A UPS can help guard against short power cuts bringing the server down.

These options can be combined to complement each other. The exact nature of individual implementation server configurations will vary depending on requirements. A recommended approach would be to use fail-over clustering and configure the disks as described in Section 1.4.

Day to day administration of SQL Server running on a cluster is no different to administering SQL Server running on a single node. The main differences are in the areas of initial configuration and licensing.

- Installing SQL Server on a cluster requires some additional steps and knowledge of the physical hardware setup in addition to knowledge of SQL Server. The installation task may therefore require database administrators and system administrators to work jointly on the task.
- Only SQL Server Enterprise Edition can be installed on a cluster server. If an active/active configuration is selected SQL Server is actually running on two separate machines it is therefore necessary to purchase a SQL Server licence for both machines in the cluster. In and active/passive configuration only one installation is running at one time therefore only one license is required.

1.10 General points about server hardware

The following points may help when specifying requirements and setting up a database server for a Portrait Foundation implementation.

- The server on which the database resides should be dedicated to this task alone. Do not additionally use the server as a domain controller, file server, print server and so on.
- Making a tape backup device available to the server may assist with the backup and recovery strategy—particularly if disk space is at a premium.
- The database server should have fast, high bandwidth links to servers in the middle tier to assist with performance.

1.11 Scaling

If the demand on a Portrait Foundation implementation grows, it is probable that it will be necessary to scale up the hardware to meet the demands.

In terms of pure database administration the simplest option by far is to scale up the hardware on which the database resides. Increasing the amount of memory, the number and speed of the processors and/or the amount of disk space available can help the performance of the database. However, as more of these

resources are added, the amount of return on the investment begins to decrease. The relationship between quality of hardware and performance is not linear.

Portrait Foundation does not currently support scaling out other than by way of the transient database. Please refer to the *Database Setup Guide* for details of the transient database.

1.12 Scheduled jobs

1.12.1 Overview

When the Portrait Foundation database is installed, a number of scheduled jobs are created to assist with the running of the system. This section describes the purpose of each of them and explains which parameters may be adjusted.

All of these jobs are given names that are appended with the name of the database. Hence, multiple databases can be maintained on one installation of SQL Server. All jobs are created in a new job category of 'Portrait' to distinguish them from other non-Portrait jobs.

These can be viewed from SQL Server Management Studio by navigating to the **Sql Server Agent/Jobs** folder. This is illustrated in Figure 6.

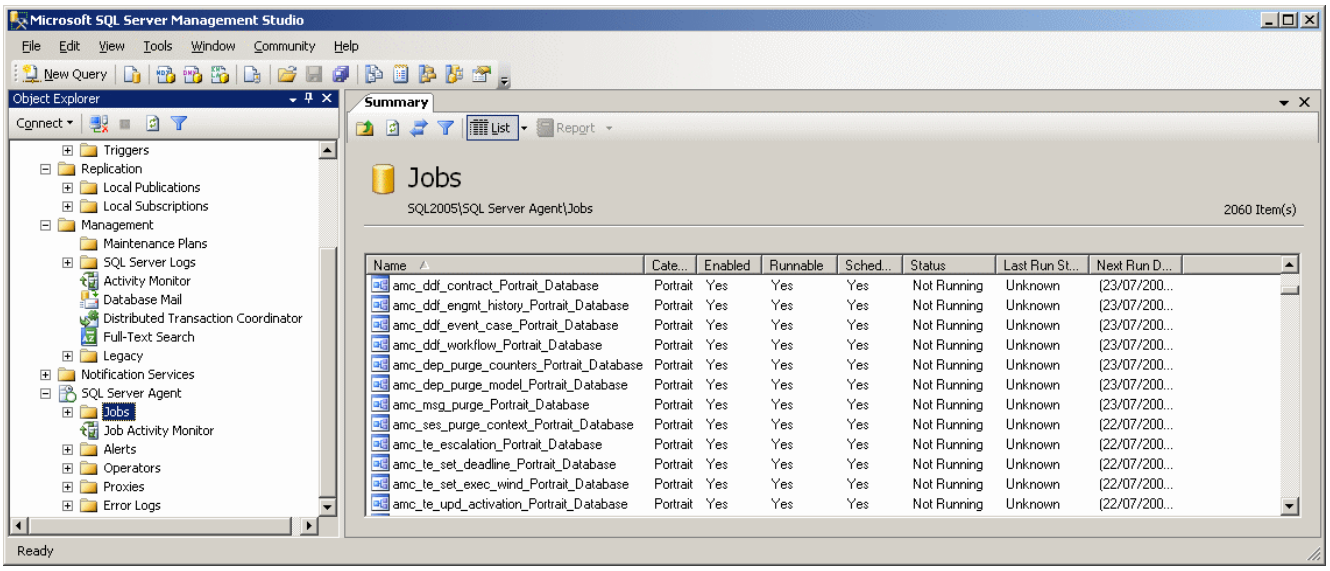
It is possible to tailor the characteristics of each job by right-clicking the job and selecting the **Properties** option. A job execution history is maintained that can be used to assess the duration and completion status of each job and its steps.

Alteration of these schedules is unnecessary in development and test environments

The job schedule may be changed to suit the requirements of specific production environments, although this would not normally be necessary. It is also possible to implement success or failure notification. Note that the scheduling of the **amc_te_escalation_<DATABASE_NAME>** job should not be altered from the default value of hourly and that the minimum period between executions for any other job is fifteen minutes.

Jobs with 'ddf' in their name refer to the Data Deletion Framework (see section 2.2 for further details of this function).

Figure 6 - Viewing the scheduled jobs



1.12.2 amc_ddf_contract

The purpose of this job is to purge old contracts from the database according to predefined deletion rules, thus reducing the size of the database and improving performance. Deletion rules are set up via the Data Deletion Framework – please refer to section 2.2 for further details.

By default the **amc_ddf_contract** job is scheduled to run every night at midnight – it calls the **p_amc_ddf_contract** stored procedure. The frequency of the job can be controlled through the SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the execution frequency to less than fifteen minutes is not a supported action. The procedure has no parameters that can be changed.

The deletion of contract data using this job will typically achieve 15,000 rows per minute. These timings were achieved using a 4 processor server with 4Gb of memory. The job has no impact on user response times.

1.12.3 amc_ddf_engmt_history

The purpose of this job is to purge old engagement actions and engagements from the database according to predefined deletion rules, thus reducing the size of the database and improving performance. Deletion rules are set up via the Data Deletion Framework – please refer to section 2.2 for further details.

By default the **amc_ddf_engmt_history** job is scheduled to run every night at midnight – it calls the **p_amc_ddf_engmt_action** and **p_amc_ddf_engagement** stored procedures. The frequency of the job can be controlled through the SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the execution frequency to less than fifteen minutes is not a supported action. The procedure has no parameters that can be changed.

The deletion of engagement history data using this job will typically achieve 15,000 rows per minute. These timings were achieved using a 4 processor server with 4Gb of memory. The job has no impact on user response times.

1.12.4 amc_ddf_event_case

The purpose of this job is to purge old cases and events from the database according to predefined deletion rules, thus reducing the size of the database and improving performance. Deletion rules are set up via the Data Deletion Framework – please refer to section 2.2 for further details.

By default this job is scheduled to run at 1am every morning. It calls the **p_amc_ddf_event** and **p_amc_ddf_case** stored procedures. The frequency of the job can be controlled through SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the execution frequency to less than fifteen minutes is not a supported action. The procedure has no parameters that can be changed.

The deletion of case or event data using this job will typically achieve 50,000 rows per minute. These timings were achieved using a 4 processor server with 4Gb of memory. The job has no impact on user response times. The job has no impact on user response times.

1.12.5 amc_ddf_workflow

The purpose of this job is to purge old workflow tasks from the database according to predefined deletion rules, thus reducing the size of the database and improving performance. Deletion rules are set up via the Data Deletion Framework – please refer to section 2.2 for further details.

By default this job is scheduled to run at 2am every morning. It calls the **p_amc_ddf_task_instance** stored procedure. The frequency of the job can be controlled through SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the execution frequency to less than fifteen minutes is not a supported action. The procedure has no parameters that can be changed.

The deletion of workflow data using this job will typically achieve 6,000 rows per minute. These timings were achieved using a 4 processor server with 4Gb of memory. The job has no impact on user response times.

Once this step has run, it then runs the task history purge routine. This stored procedure routine was introduced into databases from version 4.4 update 5 and can be retro-fitted to updated databases by running the *amc_utl_create_scheduled_jobs.sql* file which can be found on the Release CD in the Software\Tools\Database\useful_scripts folder.

⇒ If you do not want this behaviour, be sure to amend this SQL Agent Job.

The default values dictate that task history rows created more than 365 days ago will be purged from the database's *amc_te_task_history* table using a looping batch deletion mechanism. This default behaviour can be configured using the stored procedure parameters below.

Parameter name	Description
`@p_max_age_in_days`	The oldest remaining rows will be this old (default = 365 days)
`@p_batch_size`	How many records to process witin each iteration of a loop (default = 10,000)
`@p_max_iterations`	The maximum number of iterations to undertake, or NULL for unlimited (default = NULL)
`@p_disable_trigger`	To improve performance, set this to 1 if you do not use incremental DataMart updates (default = 0)
`@p_report_progress`	If you are calling this SP manually, set this to be non-zero to report progress.

Be aware that although this default behaviour on a **new** database will prevent any records in the *amc_te_task_history* table from being more than 1 year old, an updated database does *not* have this second step automatically added as a result of the database upgrade.

1.12.6 amc_dep_purge_counters

The purpose of this job is to purge old model and node counters from the database thus reducing the size of the database and improving performance.

By default this job is scheduled to run every night at midnight. It calls the **p_amc_dep_purge_counters** stored procedure. The frequency of the job can be controlled through SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the execution frequency to less than fifteen minutes is not a supported action.

Exactly what constitutes an old model is defined by a parameter to the procedure (@p_hours_old). By default this is set to 720 hours; that is, model and node counters that are older than 30 days will be purged. Implementations may change this as required.

1.12.7 amc_dep_purge_model

By default the **amc_dep_purge_model** job is scheduled to run every night at midnight – it calls the **p_amc_dep_purge_model** procedure. The frequency of the job running can be controlled through the SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the execution frequency to less than fifteen minutes is not a supported action. The procedure has three parameters:

- @p_batch_size – the number of records to process each iteration (default is 10000)
- @p_max_iterations – the maximum number of iterations, or NULL for unlimited (default is NULL)
- @p_report_progress - non-zero to report progress for each iteration (default is 0)

The purpose of this job is to purge completed models from the database thus reducing the size of the database and aiding performance. It executes on batches of n models at a time, to facilitate catching up if the number of records is so large

that otherwise the stored procedure makes no progress in a reasonable amount of time.

Models and their child nodes have a limited useful life—once the model completes there is no runtime need for the model to exist. There may, however, be a need to conduct model diagnostics post-completion.

Consequently, models have a post-completion lifespan which can be defined by model type. This lifespan is represented by the purge_interval—once the model has been complete for longer than its purge_interval then it is a candidate for being purged via this job.

The purge_interval is configurable by model type through the Portrait Management Console.

The default purge_intervals are listed in Table 2

Table 2 – Default purge_interval by model type

Model Type	Default purge_interval (hours)
Business Operation Model	24
Task Execution Model	24
Message Model	24
Data Access Model	24
Workflow Process Model	720
Application Framework Model	720
Simplex Channel Inbound Processing Model	24
Campaign Model	720
Target Selection Model	720
Generated Interaction Action Model	24
Any other model type	24

1.12.8 amc_msg_purge

By default the **amc_msg_purge** job is scheduled to run every night at midnight – it calls the **p_amc_msg_purge** procedure. The frequency of the job running can be controlled through the SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the execution frequency to less than fifteen minutes is not a supported action. The procedure has no parameters that can be changed.

The purpose of this job is to purge messages from the database thus reducing the size of the database and aiding performance. Messages that are more than 10 minutes past their timeout setting are deleted.

1.12.9 amc_ses_purge_context

The purpose of this job is to clear redundant records from the **amc_ses_session_context** table by calling the procedure **p_amc_ses_purge_context**. The table contains state information for sessions that are running within the Portrait Foundation system. The data is only relevant for the duration of a session and once a session has ended the data is no longer required. This scheduled job periodically deletes that data to ensure that disk space is not wasted.

By default the job is scheduled to run every hour and will delete records in the table that are more than 48 hours old.

The frequency of the job running can be controlled through the SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the

execution frequency to less than fifteen minutes is not a supported action. The number of hours old that a record must be before it is deleted can be adjusted by modifying the parameter passed to the stored procedure; again do this in Management Studio. If no age is specified, every record that has not been accessed for a length of time greater than the duration of its own individual purge interval is deleted. (The purge interval is configurable through the Portrait Management Console.)

1.12.10 amc_te_escalation

Workflow tasks can be configured to have their priority periodically escalated. If a workflow task has a status of active and a next escalation date in the past then the procedure **p_amc_te_escalation** that is called by this job will act on it to increase the priority.

The Portrait Foundation installation process schedules the job to run every hour. The procedure **p_amc_te_escalation** relies on the job being run hourly so it is important that the scheduling is not modified. No parameters are necessary for the stored procedure itself.

1.12.11 amc_te_set_deadline

This job raises the priority of current task instances when their deadline approaches. It does this by calling the procedure **p_amc_te_set_deadline_warning**. The database is checked for active task instances which have now passed their deadline warning date/time; the appropriate priority flag is then set for these tasks.

By default this job is scheduled to run hourly. The frequency of the job running can be controlled through the SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the execution frequency to less than fifteen minutes is not a supported action.

1.12.12 amc_te_set_exec_wind

This job sets execution windows on current task instances by calling the procedure **p_amc_te_set_exec_wind**. The database is checked for execution windows that should currently be open (the current date/time is between the start and end date/times of the execution window) and the appropriate date/times are set on the task instance.

By default this job is scheduled to run hourly. The frequency of the job running can be controlled through the SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the execution frequency to less than fifteen minutes is not a supported action. The procedure has no parameters that can be changed.

1.12.13 amc_te_upd_activation

Workflow tasks can be created (or suspended) with an activation (or reactivation) date in the future. Once the activation date has been reached, this scheduled job will call the procedure **p_amc_te_upd_activation** which updates a relevant task's status to active.

By default the job is scheduled to run every hour; no parameters are necessary for the stored procedure itself.

The frequency of the job running can be controlled through the SQL Server Management Studio. The minimum frequency is fifteen minutes. Setting the execution frequency to less than fifteen minutes is not a supported action.

1.12.14 Restored databases

Please note that the information in this section is unlikely to be relevant for production environments (a.k.a. live environments) and is aimed at development and test environments instead.

Development and test teams regularly use SQL Server Management Studio to restore database from backups but find that the Portrait Foundation scheduled job set is not present in the restored database. This is because SQL Server considers scheduled jobs to be server level objects and not database objects. Thus a database backup doesn't backup scheduled jobs as they are not considered part of the database - consequently a database restore cannot restore them. If you use the database setup wizard to "Restore from backup", the Portrait Foundation scheduled jobs should be created. See the *Database Setup Guide* for further details.

The lack of scheduled jobs means that background administration tasks do not take place and, more crucially, workflow will fail to operate correctly.

There are two approaches to overcome this problem

1. Manually create the scheduled jobs on the restored database and run them via the schedule or manually through SQL Server Management Studio and/or
2. Simulate the running of the scheduled jobs and run them when needed.

(1) Creating the scheduled jobs

The scheduled jobs can be created via a simple script. The script is called amc_utl_create_scheduled_jobs.sql and it can be found on the CD-ROM in Software\Tools\Database\useful_scripts. Please follow the simple instructions at the top of the script.

(2) Simulating the scheduled jobs

Each scheduled job has two key components – the schedule and the stored procedure(s) that it runs. The stored procedures will be present in a restored database even though the job itself is missing.

In order to simulate the scheduled job it is thus possible to simply run the underlying stored procedure on the relevant database via Management Studio. Table 3 gives details of the jobs, their procedures and the syntax.

It is envisaged that development and test teams are most likely to run the four workflow jobs in a simulated manner. Failing to simulate the data maintenance jobs is unlikely to cause any problems in low volume (ie development and test) environments.

To obtain the correct workflow function, the 4 workflow jobs must be simulated. To do this execute the following SQL using Query Analyser on a regular basis

Exec p_amc_te_escalation

Exec p_amc_te_set_exec_wind

Exec p_amc_te_set_deadline_waning

Exec p_amc_te_upd_activation

Table 3 – Simulating the scheduled jobs

Job	Purpose	Stored procedure	Example command
amc_te_escalation	Workflow (priority escalation)	p_amc_te_escalation	Exec p_amc_te_escalation
amc_te_set_exec_wind	Workflow (execution windows)	p_amc_te_set_exec_wind	Exec p_amc_te_set_exec_wind
amc_te_set_deadline	Workflow (deadline notification)	p_amc_te_set_deadline_warning	Exec p_amc_te_set_deadline_warning
amc_te_upd_activation	Workflow (task activation)	p_amc_te_upd_activation	Exec p_amc_te_upd_activation
amc_ses_purge_context	Data maintenance (session mgt)	p_amc_ses_purge_context	Exec p_amc_ses_purge_context 48
amc_dep_purge_model	Data maintenance (models)	p_amc_dep_purge_model	Exec p_amc_dep_purge_model
amc_ddf_engmt_history	Data maintenance (engmt history)	p_amc_ddf_engmt_action & p_amc_ddf_engagement	Exec p_amc_ddf_engmt_action Exec p_amc_ddf_engagement
amc_ddf_event_case	Data maintenance (case & events)	p_amc_ddf_case & p_amc_ddf_event	Exec p_amc_ddf_case Exec p_amc_ddf_event
amc_ddf_workflow (1)	Data maintenance (workflow)	p_amc_ddf_task_instance	Exec p_amc_ddf_task_instance
amc_ddf_workflow (2)	Data maintenance (workflow)	p_amc_te_purge_task_history	Exec p_amc_te_purge_task_history

1.13 Environment selection

The Portrait Foundation database contains an environment setting which helps govern the configuration and deployment change control that can be undertaken.

The environment setting is selected at database creation time and the default is 'Production'.

Production environments (often referred to as live environments) are subject to stricter configuration change control than non-production environments. Thus it is essential that databases that form part of the production system have the environment setting 'production'. Other environments that are used as part of the development and test process may be marked as 'non-production'.

Once created, the database environment type can easily be changed via the stored procedure p_amc_adm_set_environment.

e.g. exec p_amc_adm_set_environment 'Production'

or exec p_amc_adm_set_environment 'NonProduction'

Note that the environment setting affects configuration change control and deployment properties and thus running this procedure should be a carefully considered action.

Further details on the functional differences between production and non-production environments can be found in the *Live Updates User Guide.*

2 Data maintenance

As an implementation matures, there will be a need to archive or delete old and redundant data.

The Portrait Foundation database currently has facilities to purge data in a number of ways.

1. Session context – This data has a very short useful life and is purged regularly via the **amc_ses_session_context** scheduled job. Please refer to section 1.12.2 for further details.
2. Models and nodes – This data has a very short useful life and is purged regularly via the **amc_dep_purge_model** scheduled job. Please refer to section 1.12.7 for further details.
3. Configurable column reclaim – This deletion mechanism aims to reclaim configurable columns for reuse. It is invoked manually via a utility procedure **p_amc_utl_reclaim_cbe_columns**. Please refer to section 2.1 for further details.
4. Certain other entities are purged via the data deletion framework. Deletion rules are added to the system and scheduled jobs purge the data that the deletion rules declare as unwanted. Please refer to section 2.2 for further details.

2.1 Configurable column reclaim

Portrait Foundation offers users the ability to configure business entities to suit their implementation requirements without having to change the underlying physical data model.

There are five configurable entity types (party, product, repeating attributes, significant events and engagement actions) each of which can support 25 string, numeric and reference data attributes and 15 date attributes.

Occasionally these limits will be reached and users may want to add more attributes. It is not possible to add any new columns to the underlying tables and thus existing columns have to be reclaimed.

Portrait Foundation determines which columns can be reclaimed by assessing which columns, that have been used in previous deployments of the system, are unused in the latest deployment.

The reclaim is invoked manually via a stored procedure **p_amc_utl_reclaim_cbe_columns**.

For example to reclaim columns for the **Agent** party type use

```
Exec p_amc_utl_reclaim_cbe_columns
@p_cbe_type_rdg_system_name = 'PartyTypes',
@p_cbe_type_rdi_system_name = 'Agent'
```

Note that running this procedure will remove any reclaimable attribute definitions from the database and the associated business data. Hence running this procedure should be a carefully considered action.

2.2 Data deletion framework

The data deletion framework (DDF) is a broad mechanism which provides the ability to hold only the volume of Portrait Foundation data that a customer actually requires. Reasons for deletion include limited operational storage, deterioration in operational performance and legal compliance.

Note that this framework only applies to the Portrait Foundation operational database – data held in other host repositories and also in the Portrait Foundation data mart is not handled by this framework.

Currently the framework has only been implemented for Contracts, Engagements, Engagement Actions, Events, Cases and Workflow. That is, the data deletion framework only operates on these and associated child tables and does so only via specific scheduled jobs (see section 1.12).

In addition to the objects supported by the DDF the Portrait Foundation Database provides a set of generated stored procedures for deleting instances of all Portrait Foundation Entities. Customers can make use of the following stored procedures to create their own scheduled jobs.

- p_amc_del_blb_blob_data
- p_amc_del_cdo_instance
- p_amc_del_pce_case
- p_amc_del_pce_contract
- p_amc_del_pce_engagement
- p_amc_del_pce_engagement_action
- p_amc_del_pce_event
- p_amc_del_pce_milestone
- p_amc_del_pce_party
- p_amc_del_pce_proddftn
- p_amc_del_pce_repeating_atrb
- p_amc_del_te_task_instance

2.2.1 Key concepts

The key concepts of the framework are listed in Table 4.

Table 4 – Key concepts

Concept	Description
Primary entity group (PEG)	A system defined category of Primary Entity e.g. Parties, Contracts. Implementations cannot add their own groups although the default deletion rules for each group are configurable. Alternatively a PEG can be a task definition that is treated in much the same way.
Primary entity item(PEI)	A primary entity item (PEI) belongs to a primary entity group (PEG). In essence, PEIs are the logical business entities e.g. consumer. Each item may have its own deletion rules and in their absence, the item inherits the default rules of the group.
Deletion rule	The business rule for deleting data; there are two dimensions 1 Deletion start point - A configurable dimension used to determine which attribute to measure records against i.e. created when, updated when 2 Deletion period –A configurable dimension after which data is deleted. A deletion period of -1 indicated that the entity will never be deleted.

Concept	Description
Scheduling groups	A scheduling group is represented by one database scheduled job – each scheduling group may act on several PEGs or tasks and thus call several different stored procedures. These groups are not configurable – they are system defined. E.g Engagements and Engagement actions are scheduled to run consecutively as individual job steps within one job - amc_ddf_engmt_history.
Implementation hooks	Hooks that can be used to incorporate project specific functionality and processing. These are implementation specific stored procedures designed and built by the implementation teams using the naming convention below to refer to and modify the selection of candidates for deletion. See 'worked example' below.
Relationships	Most data relationships are explicitly expressed via foreign key constraints and thus these constraints prevent referential integrity failures. Other relationships are not expressed quite so clearly, notably with soft foreign keys. (when an attribute on a configurable business entity has been configured to store the primary key value of another entity, thus creating a relationship). Therefore data deletion of relationships declared via soft foreign keys must be handled by the implementation hook procedures. The product data deletion components will make no reference to them.

2.2.2 Recommendations

Whilst a deletion rule may use any date related field as its 'deletion start point', certain fields are recommended for performance reasons. These recommended deletion start points are listed in Table 5.

Table 5 – Recommended start points

Business entity	Table	Field
Engagement history	Amc_pce_engagement	Start_date_time
	Amc_pce_engagement_action	Start_date_time
Cases	Amc_pce_case	Expiry_date
Events	Amc_pce_event	Event_date_time
Workflow tasks	Amc_te_task_instance	Expiry_date
Contracts	Amc_pce_contract	Contract_data_updated_when[1]

2.2.3 Deletion mechanism

The basic deletion process operates by scheduling group with each scheduling group acting on one or many PEGs – this means each scheduled job may call one or many PEG stored procedures

For each PEG, the process will be :-

Run a 'PEG procedure' that works out the 'deletion candidates' based on the deletion rules. These candidates must exclude any records that have child records in other PEGs. Store the candidates in a temporary table of a specified name

Call an implementation 'hook procedure' of a specified name. The hook procedure will contain implementation specific processing to further validate the deletion candidates. The hook procedure will read the temporary table and confirm which candidates can be deleted ('confirmed candidates') – those rows which aren't deletable must have their is_deletable column set to 0.

1 Note that this field represents the last time any of the amc_pce_contract_data information was changed. It is maintained by a database trigger that fires when the amc_pce_contract_data table is updated.

For each confirmed candidate, call a row deletion procedure to perform the basic deletion mechanism across the physical schema.

2.2.4 Naming conventions

In order for the data deletion framework to work properly, system defined naming conventions must be adhered to. The names used currently are shown in Table 6. Future extensions will adhere to this convention.

Table 6 – Naming conventions

Product Table	Temporary table	PEG Procedure	Implementation hook procedure	Row Deletion Procedure
amc_pce_engagement	#t_amc_ddf_engagement	p_amc_ddf_engagement	p_imp_ddf_engagement	p_amc_del_pce_engagement
amc_pce_engagement_action	#t_amc_ddf_engmt_action	p_amc_ddf_engagement_action	p_imp_ddf_engmt_action	p_amc_del_pce_engagement_actio
amc_pce_event	#t_amc_ddf_events	p_amc_ddf_event	p_imp_ddf_event	p_amc_del_pce_event
amc_pce_case	#t_amc_ddf_case	p_amc_ddf_case	p_imp_ddf_case	p_amc_del_pce_case
amc_te_task_instance	#t_amc_ddf_te_task_instance	p_amc_ddf_task_instance	p_imp_ddf_task_instance	p_amc_del_te_task_instance
amc_pce_contract	#t_amc_ddf_contract	p_amc_ddf_contract	p_imp_ddf_contract	p_amc_del_pce_contract

2.2.5 Temporary table structures

Implementation hook procedures will need to use the temporary tables created by the product procedures in order to determine what the 'confirmed candidates' are. Table 7 shows the structure of these tables.

Table 7 – Temporary table structure

Entity type	Temp table name	Column name	Data type	Mandatory?
Engagements	#t_amc_ddf_engagement	Engagement_id	INT	Yes
		Engagement_type_rdg	INT	Yes
		Engagement_type_rdi	INT	Yes
		Is_deletable	NUM(1)	Yes
Engagement action	#t_amc_ddf_engmt_action	Engagement_action_id	INT	Yes
		Engagement_action_type_rdg	INT	Yes
		Engagement_action_type_rdi	INT	Yes
		Is_deletable	NUM(1)	Yes
Case	#t_amc_ddf_case	Case_id	INT	Yes
		Case_type_rdg	INT	Yes
		Case_type_rdi	INT	Yes
		Is_deletable	NUM(1)	Yes
Event	#t_amc_ddf_event	Event_id	INT	Yes
		Event_type_rdg	INT	Yes
		Event_type_rdi	INT	Yes
		Is_deletable	NUM(1)	Yes
Task instance	#t_amc_ddf_task_instance	Task_instance_id	INT	Yes

Entity type	Temp table name	Column name	Data type	Mandatory?
		Task_status_rdg	INT	Yes
		Task_status_rdi	INT	Yes
		Task_definition_id	INT	Yes
		Is_deletable	NUM(1)	Yes
Contract	#t_amc_ddf_contract	contract_id	INT	Yes
		product_rdg	INT	Yes
		product_rdi	INT	Yes
		Is_deletable	NUM(1)	Yes

2.2.6 Getting started

In order to implement the required element of the data deletion framework, a project must undertake certain tasks.

1. Determine the business rules for the deletion of data - each rule has a start_point and deletion period dimensions. The start point identifies a date column with which to determine the age of the data using the specified period, in days.
2. Add rules to the database. Three stored procedures (p_amc_ddf_save_rule, p_amc_ddf_delete_rule and p_amc_ddf_get_rules) are available to assist with this task.
3. An implementation hook procedure may be written to conduct additional processing. This may include complex business rules that can't be expressed by the dimensions of start_point and deletion_period and the handling of any soft foreign key relationships.

2.2.7 Worked example

Consider the business scenario shown in Table 8. Rules have been specified for the deletion of engagement, engagement action and workflow data - some of the rules are specific to certain entity types other are the default rules.

Table 8 – Example business rules

Business Entity	Type	Deletion period (days)	Start point	Explanation
Engagement	Default rule	90 days	Start_date_time	Remove all engagements and associated child records that were started 90 days or more ago
	Inbound call	60 days	Start_date_time	Remove all 'Inbound Call' engagements created 60 days or more ago
	Face-to-face	-1	N/A	Never remove any 'Face-to-Face' engagements
Engagement action	Default rule	90 days	Start_date_time	Remove all engagement actions and associated child records that were started 90 days ago or more
	Mortgage quote	45 days*	Created_when	Delete when 45 or more days old
Workflow	Default rule	365	Expiry_date	Default rule to delete all workflow of I year or older
	Process Complaint	-1	NA	Never delete workflows associated with customer complaints

* Mortgage quote is used as a soft foreign key elsewhere in the system.

2.2.7.1 Adding/updating rules

Rules may be added or updated via a stored procedure p_amc_ddf_save_rule.

```
exec p_amc_ddf_save_rule
@p_entity_rdg_sysname        =    <<rdg system name>>
    @p_entity_rdi_sysnme     =    <<rdi system name>>
    @p_task_dftn_sysname     =    << task definition system name>>
    @p_deletion_start_point  =    <<datetime column>>
    @p_deletion_period       =    <<number of days>>
```

Note that :-

1. A @p_deletion_period of '-1' can be used to specify a 'never delete' rule to prevent data from being removed.
2. Default rules for PEG group are setup by @p_entity_rdi_sysnme = NULL.
3. Default rules for tasks can be set by using @p_task_dftn_sysname = 'DefaultTaskDefinitionForDDF'.

For example, to create the business rules specified in Table 8 the following code is used :-

Engagements

```
exec p_amc_ddf_save_rule
    @p_entity_rdg_sysname    =    'EngagementTypes',
    @p_deletion_start_point  =    'start_date_time',
    @p_deletion_period       =    90

exec p_amc_ddf_save_rule
    @p_entity_rdg_sysname    =    'EngagementTypes',
    @p_entity_rdi_sysname    =    'InboundCall'
    @p_deletion_start_point  =    'Start_date_time',
    @p_deletion_period       =    60

exec p_amc_ddf_save_rule
    @p_entity_rdg_sysname    =    'EngagementTypes',
    @p_entity_rdi_sysname    =     'Face-to-Face',
    @p_deletion_period       =     -1
```

Note that the @p_deletion_start_point is irrelevant in 'never delete' rule above

Engagement actions

exec p_amc_ddf_save_rule

@p_entity_rdg_sysname	=	'EngagementActionTypes',
@p_deletion_start_point	=	'created_when',
@p_deletion_period	=	45

exec p_amc_ddf_save_rule

@p_entity_rdg_sysname	=	'EngagementActionTypes',
@p_entity_rdi_sysname	=	'MortgageQuote'
@p_deletion_start_point	=	'Updated_when'
@p_deletion_period	=	45

Workflow

exec p_amc_ddf_save_rule

@p_task_dftn_sysname	=	'DefaultTaskDefinitionForDDF'
@p_deletion_start_point	=	'Expiry_Date'
@p_deletion_period	=	365

Note that to set up a default workflow rule, @p_task_dftn_sysname must be 'DefaultTaskDefinitionForDDF'

exec p_amc_ddf_save_rule

@p_task_dftn_sysname	=	'ProcessComplaint'
@p_deletion_period	=	-1

Note @p_deletion_start_point is irrelevant in 'never delete' rule above

2.2.7.2 Removing rules

Rules may be removed from the system via a stored procedure p_amc_ddf_delete_rule – the sole parameter is the primary key on the amc_ddf_deletion_rule table (deletion_rule_id)

exec p_amc_ddf_delete_rule

@p_deletion_rule_id

2.2.7.3 Showing rules

To list the rules configured in the system, use the p_amc_ddf_get_rules procedure.

exec p_amc_ddf_get_rules

@p_entity_rdg_sysname	=	<<rdg system name>>

```
,@p_entity_rdi_sysname    =        <<rdg system name>>
,@p_task_dftn_sysname     =        <<task system name>>
```

For example :-

to return all engagement rules

```
exec p_amc_ddf_get_rules
    @p_entity_rdg_sysname    =        'EngagementTypes',
```

to return a specific engagement action rule

```
exec p_amc_ddf_get_rules
    @p_entity_rdg_sysname    =        'EngagementActionTypes'
    @p_entity_rdi_sysname    =        'MortgageQuote'
```

to return a specific workflow rule

```
exec p_amc_ddf_get_rules
    @p_task_dftn_sysname     =        'ProcessComplaint'
```

to return all rules, don't specify any parameters

```
exec p_amc_ddf_get_rules
```

2.2.7.4 Implementation hook procedures

In the example shown in Table 8 above, the 'Mortgage quote' engagement action is used a as soft foreign key elsewhere in the system. Thus an implementation hook procedure is required to process this key – in the example shown below the business rule is not to delete any mortgage quotes actions.

Note that to function correctly, both the procedure and the temporary table that it manipulates must adhere to the naming conventions detailed in Table 6.

```
CREATE PROCEDURE p_imp_ddf_engmt_action
AS
    UPDATE #t_amc_ddf_engmt_action
    SET is_deletable = 0
    WHERE primary_entity_type_rdg = 20
    AND primary_entity_type_rdi = 13
GO
```

3 Security

This section provides some advice on best practice security measures that will help prevent unauthorised access to the Portrait Foundation database.

3.1 General good practice

3.1.1 Physical security of the database server

The database server is a valuable asset both in terms of the hardware itself and the data it holds. It is strongly advised that the server is located in a secure room with restricted access so as to prevent unauthorised use.

3.1.2 Protect privileged user accounts

There are a number of possible routes into the database server that may be used by unauthorised users with the potential of damaging the server, including:

- The NT server administrator(s).
- Any SQL Server administrators or SQL database owners (for example, sa).
- Any accounts allowing access to the server via remote access software (such as PC Anywhere or Net Op Remote Control).

It is important that the passwords to these accounts are changed regularly and disclosed to as few individuals as possible.

3.2 Database user administration

In order for the middle tier components to be able to access the Portrait Foundation database it is necessary to set up user accounts on the database. The users should be set up within SQL Server Management Studio as follows:

- Use Windows authentication.
- Grant no server roles to the user.
- Only grant access to the appropriate database(s).
- Within the database(s) themselves, the user need only be granted **Public** role.

Following these guidelines will give a user sufficient database access to enable them to run Portrait Foundation. It may be necessary to grant users on development databases further privileges. It is recommended, however, that access is restricted as much as is possible.

3.3 Database object permissions

All permissions on objects within the Portrait Foundation database are managed using the **Public** role. In order to run Portrait Foundation it is not necessary to grant individual users specific privileges on specific objects. Instead users need only be granted the **Public** role.

During the installation process, once the tables, stored procedures and other objects have been created, a script is run to grant execute permission on all stored procedures to **Public**. Additionally further permissions on specific tables are granted—this is to support areas of the Portrait Foundation application that create and execute dynamic SQL and also to support stored procedures that use dynamic SQL.

4 Extending the Portrait Foundation database

The Portrait Foundation database has been designed to provide a substantial amount of flexibility by the use of generic data structures and metadata to describe the contents of these structures.

That said, there will be instances whereby the business solution requires data elements that cannot be represented in the Portrait Foundation database schema. Note that these occasions are likely to be rare and thus project based database schema changes should be a carefully considered action. Projects should contact Portrait Support for advice before undertaking any of these changes.

This section describes how to extend the Portrait Foundation database to include implementation specific objects and how to install them.

4.1 Portrait Foundation database objects

The first and last rule to be observed under this heading is that a Portrait Foundation implementation team should not modify any Portrait Foundation product database object (table, view, stored procedure, index, constraint or any other object). Product objects may be identified by their naming conventions as listed in

Table 9 – Portrait product database objects

Object	Portrait naming convention
Table	Amc_<table name>
Index	<prefix>_amc_<index name>
View	v_amc_<view name>
Procedure	p_amc_<procedure name>
Function	f_amc_<function name>
Constraints	Any foreign keys owned/operating on a Portrait table
Scheduled job	Amc_<job name>_<database name>

The Portrait Foundation database has been designed to provide a substantial amount of flexibility by the use of generic data structures and metadata to describe the contents of these structures. If an implementation has a requirement that suggests a need to modify an existing database object it is essential that the Portrait Foundation product team is contacted in order to discuss the requirement.

The reason for this restriction concerns upgrade paths. The Portrait Foundation product team will regularly release upgrades to the product. The upgrades will include scripts to modify the database structure. The scripts will assume that an implementation team has not modified the database. If unauthorised modifications are made, they are at risk of being overwritten as the upgrade is performed.

4.2 Implementation database objects

It is possible that during an implementation project it will be necessary to create additional database objects in order to support an extra piece of functionality. The process to deal with such a request will vary depending on the nature of the functionality being implemented.

Before considering the addition of a new database table, you should question whether or not a new supplementary object (utilising existing database tables) would fulfil the requirement. Creating additional tables should be considered with caution as it runs the risk of compromising existing functionality. That said, there will be occasions when the existing product functionality does not suffice and new entities will be necessary.

4.2.1 Conventions

It may be necessary to create additional database objects for an implementation to support functionality required for that implementation alone. In this case one must follow the conventions listed in Table 10:

Table 10 – Implementation specific database objects

Object	Conventions
Table	Named imp_<table name> Stored in IMPDATA filegroup
Index	Named <prefix>_imp_<index name> Stored in IMPINDEX filegroup
View	v_imp_<view name>
Procedure	p_imp_<procedure name>
Function	f_imp_<function name>
Constraints	Any constraint owned by or acting on an implementation table. Care must be taken if implementation tables have foreign keys to product tables. It is possible that modifications to the referenced product tables will take place as part of a product upgrade. The presence of non-product foreign keys could cause the upgrade to be unsuccessful; to avoid this situation it is necessary to drop these foreign keys during an upgrade.
Scheduled job	Imp_<job name>_<database name>

4.3 Installing implementation specific database objects

Once implementation objects have been designed and coded they must be installed into the Portrait Foundation database. The implementation objects will operate along side the core product objects and will be distinguished via their naming conventions (see table and install mechanism).

The Portrait Foundation database setup wizard includes 'hooks' by which implementation database scripts may be incorporated. These hooks exist for both the 'full creation' and 'upgrade' of the database – the upgrade mechanism allows implementation scripts to be run on their own without the inclusion of product scripts. This means that the same upgrade mechanism may be used to install either base product upgrades, implementation specific upgrades or both together, provided that the correct set of scripts is provided.

Please refer to the *Database Setup Guide* for installation instructions.

4.3.1 Implementation 'hook' scripts

The install hooks require the implementation files to adopt a predetermined name and directory structure in order to operate properly. Any other naming convention or directory structure is not supported.

The required file names and directory structure is listed below. Project teams should construct an identical structure in their own file system or source control system.

Db_Create\

Cre_imp_db.txt

Patch_imp_db.txt

Imp_schema.sql

Patch_imp_schema.sql

Admin\

Drop_imp_procedures.sql

Grant_imp_execute.sql

Grant_imp_tab_privs.sql

Ins_version_record.sql

Tsql\

DB_create folder

The cre_imp_db.txt and patch_imp_db.txt scripts form the backbone of the implementation database 'full creation' and 'upgrade' respectively – at install time they are converted to .bat batch files. They make calls to numerous other scripts each of which fulfils a distinct purpose. The calls are made using the SQL Server SQLCMD command line utility.

```
--This command calls the script that creates the implementation tables and indexes
 SQLCMD -S <T_SQL_Srv> -h-1 -w512 -d <T_SQL_dbname> -U <T_SQL_user> -P <T_SQL_pass>
-i <filename> -o <logfilename>
```

Each SQLCMD command uses various tags that are replaced with real values at install time. The install will expect the tags to be exactly as specified so care should be taken not to change them.

<T_SQL_Srv> = server name

<T_SQL_dbname> = database name

< T_SQL_User_Password > = username and password

Both the cre_imp_db.txt (full creation) and patch_imp_db.txt (upgrade) files make calls to numerous files. These calls and their purpose are shown in Table 11.

Table 11 - The implementation install files

Filename	Purpose	Called in full creation	Called in upgrade
Imp_schema.sql	Contains all the implementation DDL (create table, create index, create constraint). This file could be the output from a database design tool.	Y	N

Filename	Purpose	Called in full creation	Called in upgrade
Patch_imp_schema.sql	Contains the implementation DDL required for a particular database upgrade	N	Y
drop_imp_constraints.sql	Drops all the constraints owned by implementation tables	Y	N
drop_imp_tables.sql	Drops all the implementation tables	Y	N
drop_imp_procedures.sql	Drops all the implementation procedures. This script is called in both the full creation and the upgrade as the policy is always to drop and reload the entire procedure set thus ensuring the latest code is in use.	Y	Y
cre_imp_2_base_ri.sql	Creates all the foreign keys that use product tables as the 'child' reference.	Y	N
recomp_imp_procedures.sql	Recompiles all the implementation procedures thus resolving any issues due to procedures having not been created in the correct dependant order	Y	Y
grant_imp_execute.sql	Grants execute on all the implementation procedures	Y	Y
grant_imp_tab_privs.sql	Grants privileges on implementation tables	Y	Y
ins_imp_version_record.sql	Inserts the implementation version number	Y	Y
create_imp_procedures.sql	Creates the implementation procedures. The procedures in the tsql directory are collated into this one file at install time	Y	Y
create_amc_delete_procedures.sql	Creates the 'row deletion' procedures – these procedures are designed and created at install time.	Y	Y

The Tsql folder

This folder will contain a file for each implementation specific stored procedure created. The install process will concatenate all the files together and run them from one script called create_imp_procedures.sql. It is therefore important to ensure that each file has a blank line at the end of it, so that the resulting file will include spaces between the code for each procedure.

Database jobs

If Portrait Foundation projects wish to create implementation specific stored procedures which are run as database jobs, it is advisable that these jobs are created in the same way as for the Portrait Foundation product jobs.

To do this: create the job in SQL Server Management Studio, then script out the job creation to a file. Save this file in the admin directory and call it Save_job_<name of procedure>.sql. The script can then be called from the main controlling script (cre_imp_db.txt or patch_imp_db.txt). An example script call is provided in cre_imp_db.txt for guidance. The stored procedure should be called in the Tsql folder with all other procedures.

4.4 Getting started

This section lists some 'what if' scenarios to help implementations undertake database extensions. Note that this list doesn't cover all potential circumstances – it is only intended to give a the reader an idea of what may need to be undertaken when project want to add their own implementation database objects.

Amending a Portrait Foundation object

You can't! – an implementation is not permitted to amend any Portrait Foundation product object. Only implementation tables may be added or deleted. Please refer to section 4.1 for details of how to identify Portrait Foundation database product objects.

Adding an implementation table

1. Design the table as you would for any piece of database work adding columns, primary keys and foreign leys. Note the naming conventions in Table 10.
2. Add the DDL to the imp_schema.sql file and if the work is for a particular upgrade, add it to the patch_imp_schema.sql file.
3. Add the relevant grant to the grant_imp_tab_privs.sql
4. If you wish to create a new database with this table in it, run the database install on the CD-ROM. The implementation hooks will pick up the amended files and create the new table.
5. If you wish to add this table to an existing environment, run the database upgrade found on the CD-ROM. The implementation hooks will pick up the amended files and add the new table.

Amending an implementation table

1. Write the ALTER TABLE DDL and add it to the patch_imp_schema.sql file. Also make amendments to CREATE TABLE DDL in the imp_schema.sql file
2. If you wish to create a new database with this table in it, run the database install on the CD-ROM. The implementation hooks will pick up the amended file and create the new table.
3. If you wish to add this table to an existing environment, run the database upgrade found on the CD-ROM. The implementation hooks will pick up the amended file and add the new table.

Adding an implementation procedure

1. Code the new procedure as you would normally. Note the naming conventions in Table 10.
2. Add the procedure to the Tsql directory
3. Add the relevant grant to the grant_imp_execute.sql file
4. If you wish to create a new database with this procedure in it, run the database install on the CD-ROM. The implementation hooks will pick up the new file and create the new procedure.
5. If you wish to add this procedure to an existing environment, run the database upgrade found on the CD-ROM. The implementation hooks will pick up the new file and add the new procedure.

Amending an implementation procedure

1. Amend the file in the Tsql directory
2. If you wish to create a new database with this procedure in it, run the database install on the CD-ROM. The implementation hooks will pick up the amended file and create the new improved procedure.

3 If you wish to add this procedure to an existing environment, run the database upgrade found on the CD-ROM. The implementation hooks will pick up the amended file, drop the old procedure and create the new improved procedure.

Deleting an implementation procedure

1 Remove the file from the Tsql directory

2 If you wish to create a new database without this procedure in it, run the database install on the CD-ROM. The implementation hooks will not find and thus not create the procedure.

If you wish to remove this procedure from an existing environment, run the database upgrade found on the CD-ROM. The implementation hooks will not find the procedure. All procedures will then be dropped and only those found by the implementation hooks will be recreated.