Portrait
Foundation

Batch Loading Framework Configuration Guide

Edition 3.1

10 January 2013

Pitney Bowes
Software

Portrait Foundation
Batch Loading Framework Configuration Guide

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 as a guide for the configuring the Batch Load Framework for bulk loading data into the Portrait Foundation metadata driven database. Details of how to operate the Batch Load Framework at runtime are in *Batch Load Framework Operational Guide*.

Intended audience

This document is intended for users who create and maintain batch load definitions using the Batch Load Framework extensions to the Configuration Suite. It is assumed that readers will be familiar with the Configuration Suite and concepts related to the Configuration Suite.

Knowledge of the Portrait Foundation database tables would be beneficial but not assumed.

Related documents

Batch Load Framework Operational Guide

Software release

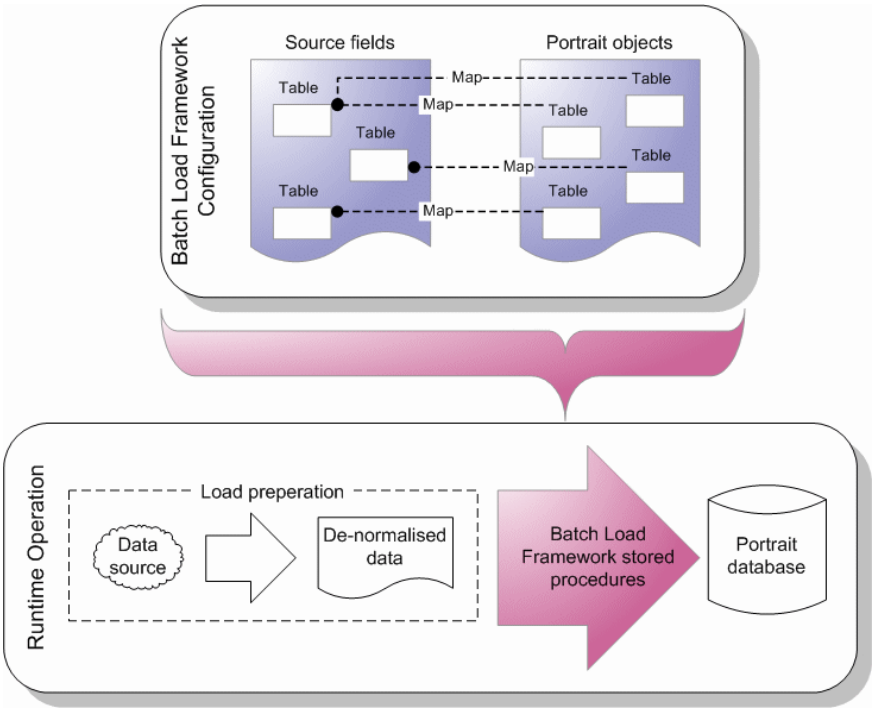
Portrait Foundation 3.2 or later.

Contents

1 Overview of the Batch Load Framework

The Batch Load Framework (BLF) provides a facility for bulk loading data into the Portrait Foundation metadata driven operational database. The BLF can be installed using the standard Configuration Suite and InstallShield scripts.

Figure 1 – Batch Load Framework architecture



The BLF process is in two distinct parts; configuration, done through the Configuration Suite, and batch-load of de-normalised data at runtime. This document is as a guide for configuration of the BLF using the Configuration Suite.

Details of how to use BLF to load de-normalised data at runtime are in the Batch Load Framework Operational Guide.

1.1 Main features

There are many features of the BLF the following are specific to configuring the BLF.

- Support for multiple load definitions.
- Extensions to Configuration Suite for defining batch loads using the new Batch Load Editor.
- Abstraction from the underlying database tables for Portrait Foundation configurable entities. There is full support for populating hierarchical objects, search and unique tables.
- Support for updates by identifying existing objects through their unique attributes.
- Support for the definition of objects that enable loading into custom implementation tables.
- Support for simple conditions to define logic for which objects are relevant in incoming rows.

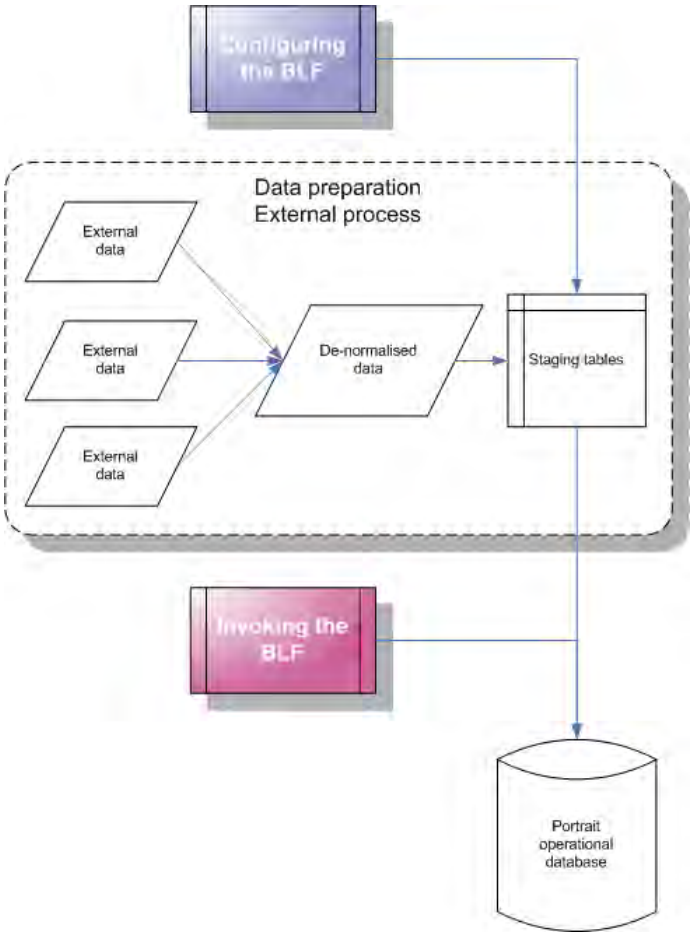
- Ability to define mutually exclusive objects that allows source data to drive which exact types are created, but simplifies configuration by treating them as a single object. This is relevant where creation of further dependent objects is need.
- Support for resolving reference data system names to IDs.

2 Batch load process

The batch loading process can be split into three parts.

1. Configuring the BLF to deploy staging tables and conversion procedures - The BLF configuration is done through the Configuration Suite and defines the relationships and transformations for the external data so that it can be placed into the Portrait Foundation database. These definitions are deployed to the database where a staging table and load procedures are stored. The staging table is a de-normalised view of all objects involved in the load and reflects the structure of the source file.
2. Preparing external data so that it can be loaded into the staging tables - Once staging tables have been created data from the data source can be loaded into them ready for the BFL procedures to load the data into the Portrait Foundation database at runtime.
3. Invoking the BLF procedures to load the staging table data into the Portrait Foundation database - The BLF procedures can be invoked, using the command line, Microsoft SQL Server Integrations Services (SSIS) or any other suitable tool, to pre-process and load the data from the staging table into Portrait Foundation database. Any errors that occur are written to an error table identifying the row that failed. In addition, auditing tables are populated for reporting on each batch instance.

Figure 2 – Batch load process



2.1 Configuring of BLF

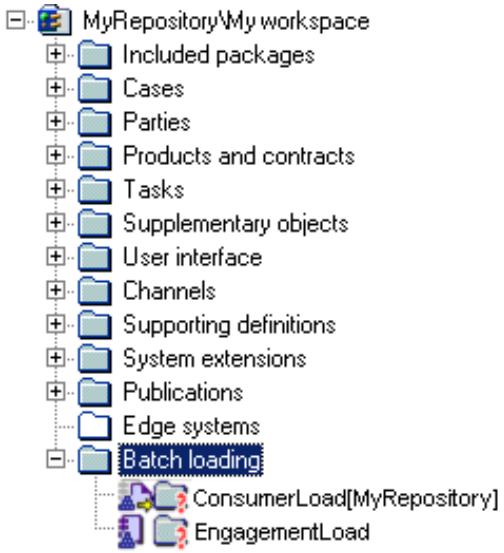
The BLF is a set of tools and database extensions that allow users to define and execute batch loads for populating the entities held in the Portrait Foundation

metadata driven operational database, typically from a de-normalised source file such as CSV text file.

2.2 Configuration Suite extensions

Batch load definitions are stored in the root of the workspace under a node called **Batch loading**.

Figure 3 – Batch loading section of the configuration tree



Definitions are created and maintained using the Batch Definition Editor.

3 Creating a batch load

The BLF needs to be configured to deploy staging tables and procedures for batch loading of data into the Portrait Foundation metadata driven operational database.

3.1 Adding BLF to the Configuration Tree

Batch load definitions are stored in the root of the Portrait Foundation Configuration Suite workspace under a node called **Batch loading**. If Batch loading is not in the configuration tree it can easily be loaded.

1 Start Configuration Suite and open a workspace.

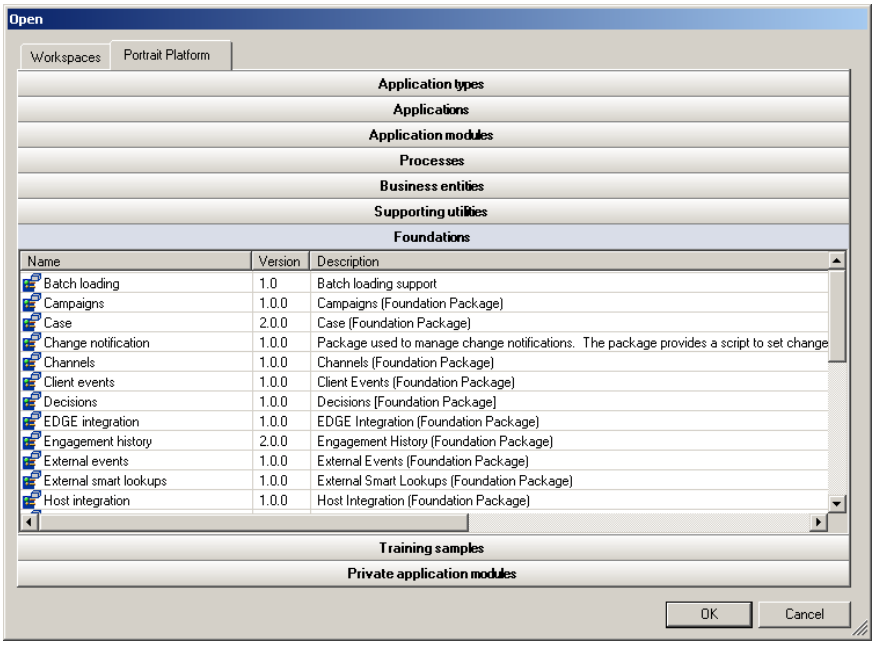
2 Click **Open** from the **File** menu.

3 Click the Portrait Platform tab.

4 Click Foundations.

5 Click the Batch loading definition.

Figure 4 – Location of the Batch loading application



6 Click **OK** to append the Batch loading definition to the end of the configuration tree.

3.2 Configuring BLF

This section steps through the creation of a simple batch load using the Portrait Foundation Configuration Suite. It is assumed that you have knowledge of the source data structure and Portrait Foundation data tables. An explanation of the Portrait Foundation database is in the *Database Administrators Guide*.

1 Right-click on Batch loading and click New batch definition.

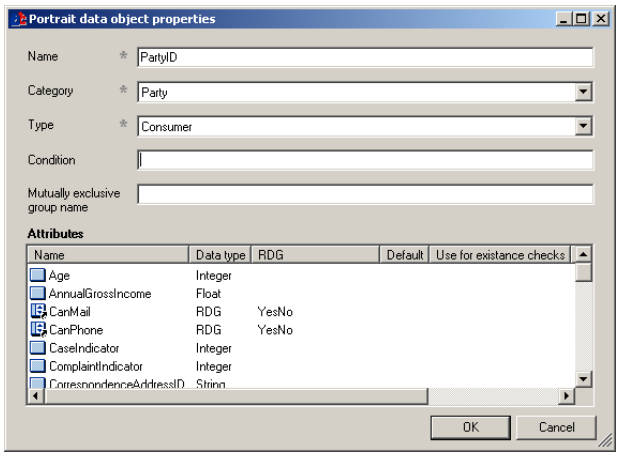
2 In the **Batch name** box, enter a suitable name for the batch load.

3 Click the Portrait object tab.

4 Click **Add** to add a Portrait data object as the destination for the batch load.

5 Complete the **Portrait data object properties** dialog and click **OK**.

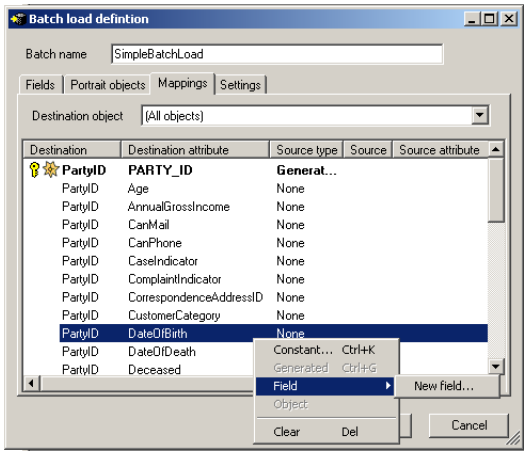
Figure 4 – Portrait data object properties



6 Click the **Mapping** tab to show the attributes of the clicked Portrait data objects.

7 Right-click on a destination attribute and click the appropriate source.

Figure 5 – Adding a new field mapping



8 Click the **Fields** tab to show details and order of the fields that will be created in the staging table.

9 Click the **Settings** tab to show details of setting for the batch load.

10 Click **OK** to save the new batch load.

3.3 Deploying the BLF

Batch load definitions are deployed in the same way as other Portrait Foundation configuration – using the configuration deployer component.

The batch load deployer writes the definitions of the fields, Portrait Foundation objects and mappings for the batch definition to the appropriate database tables.

In addition the deployment process also creates the database objects to support the load - the staging table, an error table and the load stored procedures.

The database objects that are created are summarised below:

3.3.1 Tables

Table name	Description
amc_blg_<BLName>	Staging table. This table is created from the field definitions plus some additional utility columns.
amc_blg_<BLName>_error	Error table. This table just has columns that match the field definitions. When a batch instance completes any rows that failed to load are moved to this table.

3.3.2 Stored procedures

Stored procedures are generated for inserting each Portrait Foundation object defined in the batch definition.

The procedures are generated using the prefix `p_amc_blg_` followed by the load name and the object name. i.e.:

p_amc_blg_<BLName>_<ObjectName>

For example:

p_amc_blg_ConsumerLoad_Consumer

If the Perform Updates option is selected procedures for updating objects are also generated, but with the `p_amc_blg_upd` prefix.

For example:

p_amc_blg_upd_ConsumerLoad_Consumer

In addition to the procedures for loading each object a single stored procedure for controlling the execution of all of the objects for a chunk is also created. This procedure is called by the Batch load framework at runtime to load each chunk.

The procedure is named:

p_amc_blse_<BLName>

For example:

p_amc_blse_ConsumerLoad

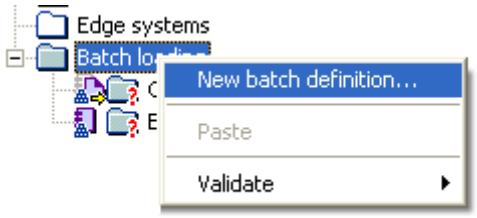
None of these procedures should be called directly, they are only designed to be used within the batch load framework.

See the document '**Batch Load Framework – Operational Guide**' for details on preparing and executing batch load definitions.

4 Batch definition settings

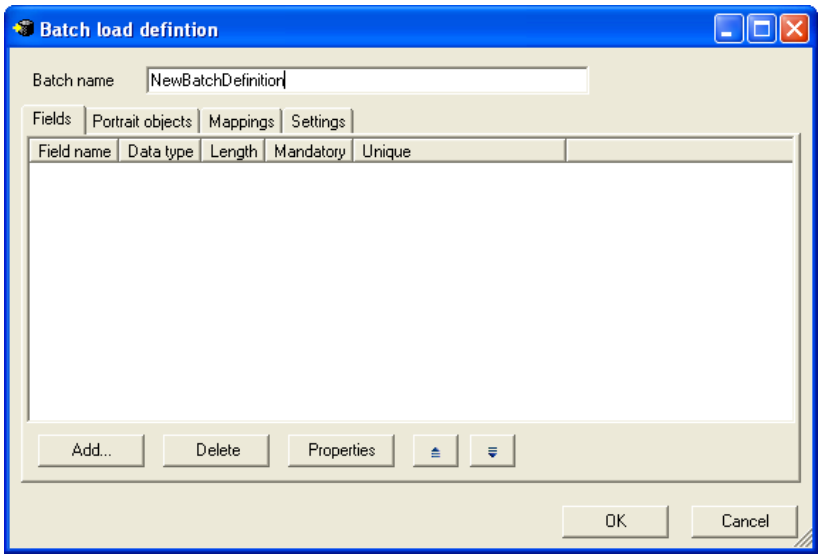
To create a new batch load definition left click on the **Batch load** node and select **New batch definition**.

Figure 6 - Creating a new batch definition



This will show the Batch load definition dialog.

Figure 7 - Batch load definition dialog



Enter a name for the batch definition in the **Batch name** field.

Batch load definition

The editor has four tabs.

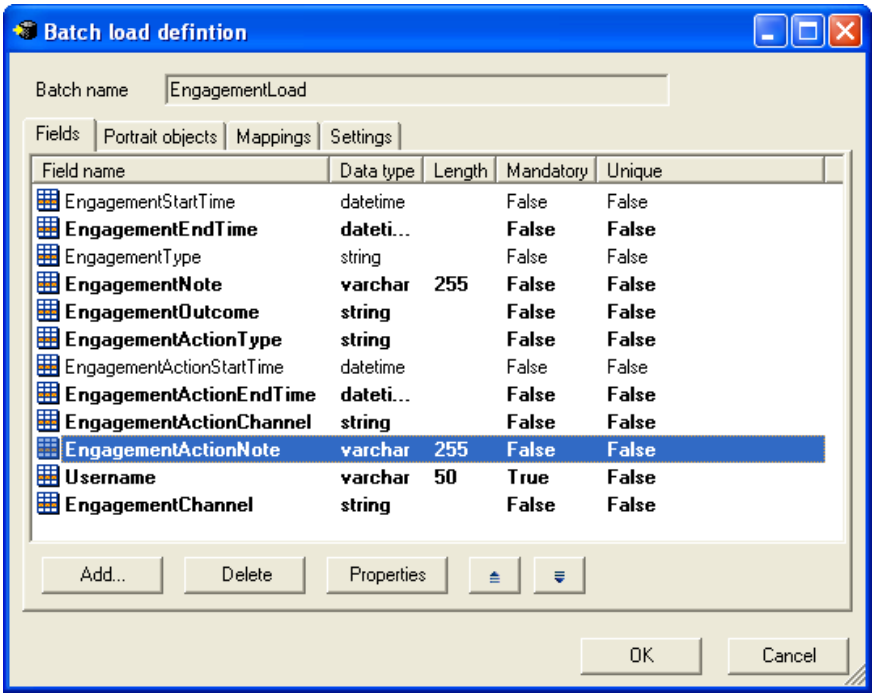
- **Fields** tab defines the fields in the source data file and creates the staging table into which the file is loaded.
- **Portrait objects** tab defines the target objects in the Portrait Foundation database that will be created.
- **Mappings** tab defines how the Portrait objects are populated from fields and constants, and create relationships between objects.
- **Settings** tab specifies settings for the particular batch load including the chunk size, use of external references and whether to perform updates on existing objects.

4.1 Field definitions

The field definitions are used to create the staging table for the batch load. Each field represents one column in the staging table.

The order the fields are displayed in the list is the order that the columns are created in the staging table.

Figure 8 - Fields tab in the Batch load definition dialog



4.1.1 Adding a new field definition

To add a new field definition while on the **Fields** tab select the **Add** button.

Figure 9 - Field definition properties dialog

Field properties

A field definition has the following properties.

- **Field name** A unique name for the field.
- **Data type** Select the data type for the field. Values supported include
  - `string` - generates `varchar(50)` columns.
  - `varchar` - must specify length.
  - `int`
  - `bigint`
  - `boolean` - generates `numeric(1)` columns.
  - `float` - generates `double precision` columns.
  - `datetime`

  If the `varchar` type is selected the Length field must be specified. Values in the range 1 – 256 are supported.

- **Mandatory** Check box if the source data must contain a non null value for this field. If this is ticked the generated table will have a NOT NULL constraint applied.
- **Unique** Check box if each source data row must contain a unique value for this field. If this is ticked the column in the generated table will have a unique constraint applied.

4.1.2 Modifying a field

To prevent invalid mapping, the name and data type cannot be modified once the field has been created. To change the **Name** or **Data type** properties, the field must be deleted and recreated.

The **Length**, **Mandatory** and **Unique** properties can be changed at any time.

- With the field to modify selected in the list box select **Properties** in the **Batch load definition** dialog.

4.1.3 Deleting a field

Select the field to delete in the **Batch load definition** dialog.

Select **Delete**, and then select **Yes** to confirm deletion of the field.

Any mappings that use the field are reset.

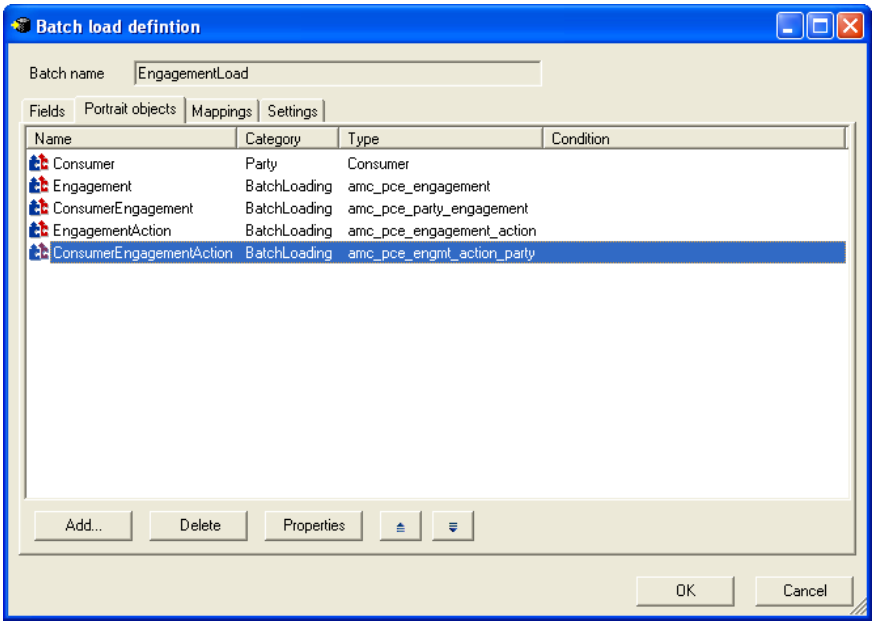
4.1.4 Ordering fields

The displayed order of fields in the list is the creation order of columns in the staging table. Change the order of fields in the list by selecting the field and using the up button ( ) and down button ( ) accordingly.

4.2 Portrait Foundation Objects

The Portrait Objects tab shows the objects that will be loaded as part of the batch load definition. The list shows the name of the object, its category and type and any conditions applied to loading the object.

Figure 10 – Portrait objects tab in the Batch load definition dialog

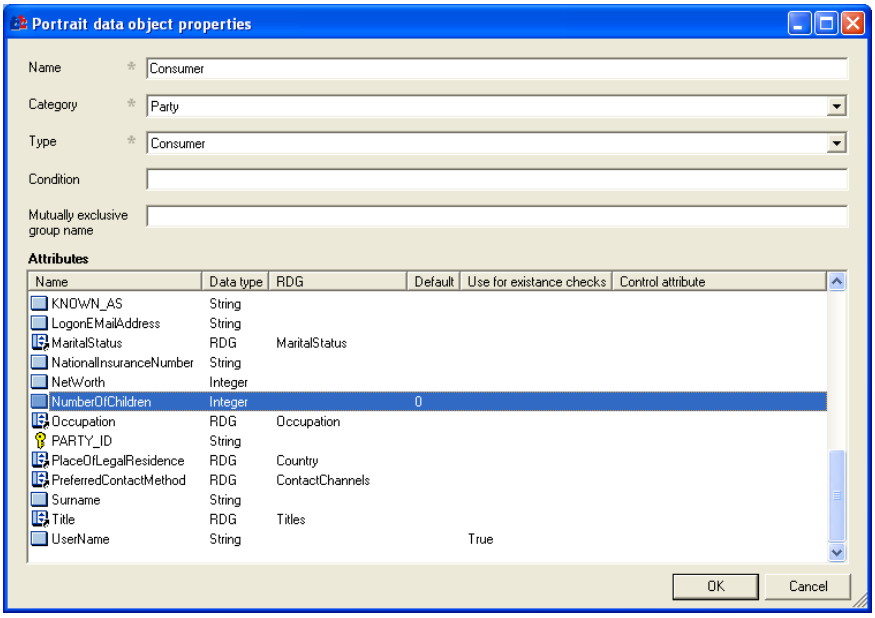


The order of the objects in the list is the order that the objects will be processed for each source data row at runtime. Therefore, if the there are dependencies

between objects then the primary object must be processed first before any dependent objects. For example where the id of a party needs to be set on a repeating attribute, the party object must be processed first. The order of the objects also restricts the mappings that are made available on the mapping page.

4.2.1 Adding a new Portrait Foundation object

To add a new Portrait Foundation object while on the **Portrait objects** tab select the **Add** button.



Enter a name for the object. The name must be unique within the batch load definition and must comply with the constraints defined for System names as they are used to generate the names of utility columns in the staging table.

Object Category and Type

Select the category of object. The categories that are supported by the batch load framework are:

- Party,
- Product,
- Engagement action,
- Significant event,
- Repeating attribute,
- Contract to contract relationship,
- Party to party relationship, and
- Batch loading

All types that have been configured for the selected category can be loaded using the batch load framework.

The **Party**, **Product**, **Engagement action**, **Significant event** , **Repeating attribute**, **Contract to contract relationship** and **Party to party relationship** category objects provide the abstraction from the underlying Portrait Foundation table structures that support these entities.

For example, Party objects are stored in the `amc_pce_party` and `amc_pce_party_type_data` tables. Each level in a party hierarchy is stored in a separate row in the `amc_pce_party_type_data` table. In addition search and unique attributes are stored in the `amc_pce_party_search` and `amc_pce_party_unique` tables respectively.

A summary of which tables are populated for each category can be found in 5.

The **batch loading** category is used to provide support for :

- creating stub records for the above object types,
- populating non-configurable Portrait Foundation objects (e.g. Engagements and campaigns)
- custom / implementation specific tables.

The following types of object are provided within the Batch loading category.

For populating stub records:

- Repeating attribute
- Engagement action
- Significant event

The creation of Party and Contract stub records is not supported as the stub table only contains the id. It could be useful to create stub records for the other entities because they contain more useful information and are not hierarchical.

For populating non-configurable Portrait Foundation objects:

- Party event
- Contract event
- Contract participant
- Engagement
- Engagement action party
- Party engagement
- A summary of the tables that these objects populate can be found in 5.

For populating custom objects:

- Any configured custom object. See Custom objects.
-
- A summary of how each of these objects are related is described in 5.2.

Conditions

The batch load framework supports simple conditions for determining at runtime which objects are relevant for a particular row in the staging table.

The condition can only take account of the values of fields in the staging table.

Therefore, the condition must take the form:

Field_name <operator> Constant_Value

e.g. CustomerType = 'Organisation'

This allows different objects to be switched on and off for a row based on values in the staging table.

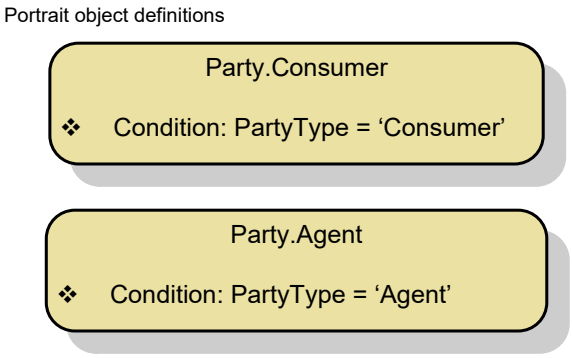
For example, if we had the following fields in the staging table:

Staging table Field definitions

PartyType	FirstName	Surname	Etc..

If two different Portrait Foundation objects (an Agent or a Consumer) should be created depending on the value of 'PartyType' two different objects should be defined and mappings made from staging table fields to appropriate attributes on each of the objects.

A condition is then defined for each object to set which object is relevant at runtime.



The objects created under different values for PartyType are :

Value of PartyType	Objects created
'Consumer'	Consumer
'Agent'	Agent
'Broker'	no objects
NULL	Both objects! Unfortunately the conditions evaluate to 'undefined' in this situation and both will be created. In order to handle null values the conditions need to be extended to check for NULL. E.g. PartyType = 'Agent' AND PartyType IS NOT NULL. Alternatively, if the value of PartyType should never be NULL the field should be marked as 'Mandatory' in the Field properties dialog.

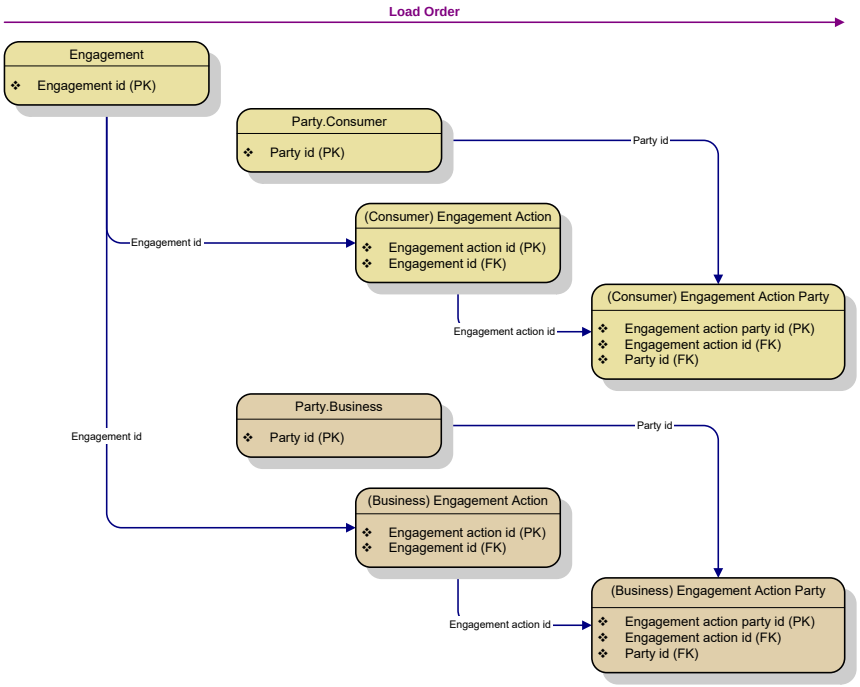
Mutually exclusive objects

In certain circumstances there may be two or more mutually exclusive objects when processing rows in the staging table. This is illustrated in the example above where a condition is used to active one of two objects depending on a value in a field in the staging table.

The number of Portrait Foundation objects that need to be defined can grow to an unmanageable size where there are dependant objects which need to be related to those objects.

For example, take the situation where, for each row in the staging table, an Engagement object, followed by either a Consumer or a Business party object, followed by an engagement action for each party are created.

Because the party id of the party needs to be mapped on to the engagement action object we need to define two engagement actions objects – one for the Consumer type and one for the Business type.

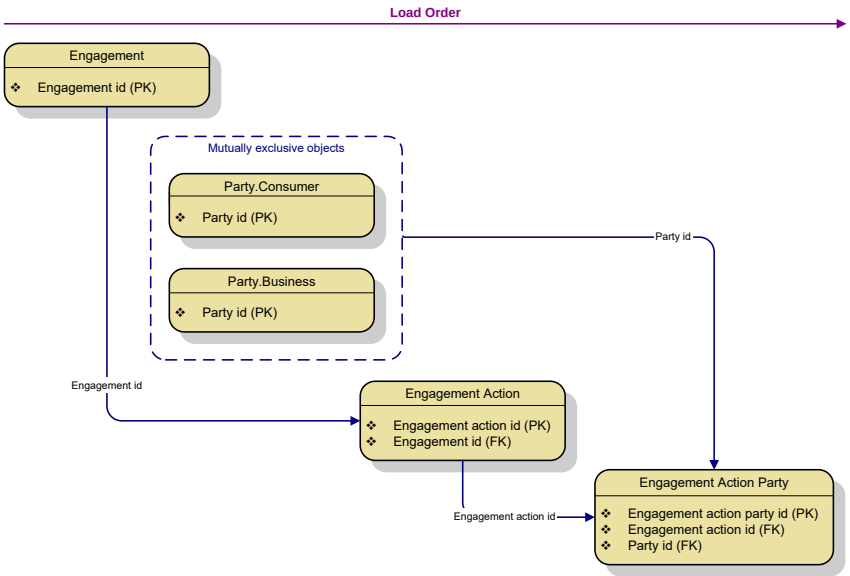


This would probably be acceptable if the complexity is at this level. However, if there are more dependencies, e.g. Party engagement objects or repeating attributes, or the number of possible party types increases then the amount of objects required to handle all permutations rapidly increases.

This makes maintenance of the load definition very difficult and can effect the runtime performance due to the number of objects that need to be checked and evaluated to see if they are relevant in each row.

The 'Mutually exclusive group name' is used to overcome this problem by grouping objects that are mutually exclusive at runtime.

Instead of duplicating all downstream objects we can group those that are mutually exclusive and map from only one of them to the destination object. The Batch load framework will work out at runtime which object is relevant and, therefore, which object to map from.



There is a restriction on which objects can be grouped in this way. Each object must target the same physical table and have the same primary key defined. So grouping objects of the same Category and Type is allowed and also objects such

as `EnagementAction.<Configured action>` and `BatchLoading.EngagementAction` can be grouped as they target the same physical table.

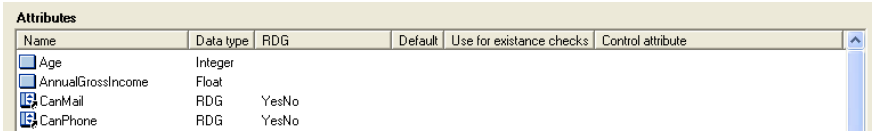
Portrait Foundation Object attributes

When a Portrait Foundation object's Category and Type have been selected the attributes of the object are shown in the list view.

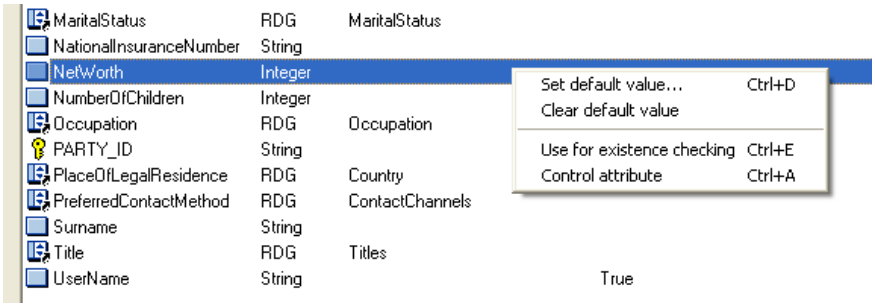
The columns in the list view show the names of the attribute, the data type, reference data group (where applicable), default value, existence checking flag and control attribute flag.

The primary key for an object is indicated with a icon.

Reference data attributes are shown with the icon, and all others with the icon.



These columns can be modified by selecting the appropriate menu option on the context menu or by using the shortcut keys shown below.



Default value

To set a default value for a Portrait Foundation object attribute select the 'Set default value...' menu option or press the short cut key Ctrl+D.

A dialog for setting the default value for the selected attribute will be displayed. The type of dialog shown depends on the data type of the attribute (Integer, String, RDG, Datetime or Boolean).

If a default value is set for a Portrait Foundation object attribute then it will be set on an object created at runtime if :

a no mapping is made to this attribute, or

b a mapping is made but at runtime the source value is null.

Existence checking attributes

This option is only available on attributes that are configured as being unique for a particular object type.

If this option is selected the attribute will be used to check for existing objects in the Portrait Foundation database.

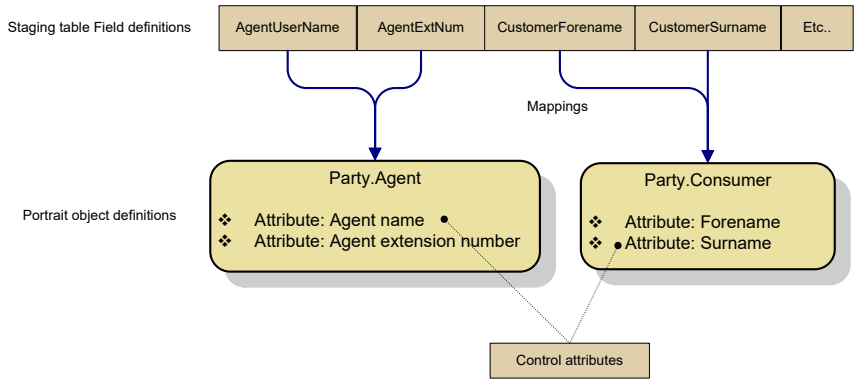
At runtime the field that is mapped to this attribute is looked up in the appropriate table (for CBE's - the 'unique' table and for custom objects - the table itself).

If the object already exists a new object will not be created. If the 'perform updates' option is selected for the batch definition then any objects that currently exist will be updated.

Control attributes

If the 'Control attribute' flag is set then the object will only be loaded if a non-NULL value is supplied for this attribute at runtime. This provides some control over when objects should be created in addition to the Condition rules defined for the object.

This is primarily used where optional or mutually exclusive objects are expected in the source file, typically where some fields are provided if one object is relevant and a separate set of fields for a different object.



In the example above the Agent object has the `AgentUserName` and `AgentExtNum` fields mapped and the Consumer object has the `CustomerForename` and `CustomerSurname` fields mapped.

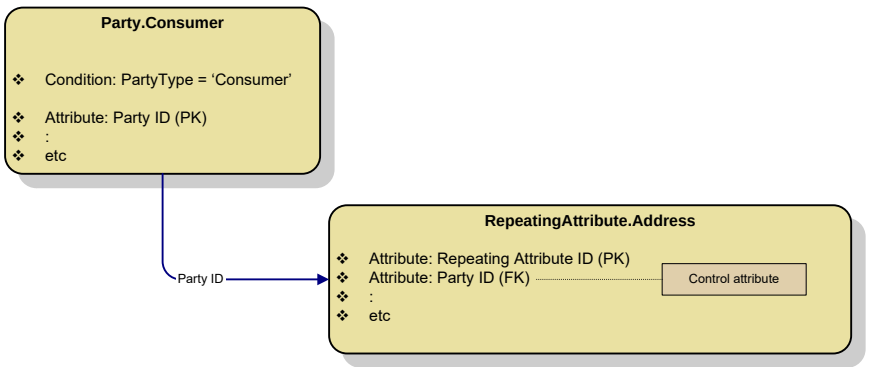
The Agent name attribute on the Agent object and the Surname attribute on the Consumer object are set as being Control attributes. Therefore, when `CustomerSurname` is NULL the Consumer object will not be created and likewise for `AgentUserName`.

It is assumed that the source file will be populated with Agent or Custom fields but not both, for each row in the source file.

Object dependencies

The flag can also be used where mappings are made from another object – Primary key / Foreign key assignment, in order to switch off objects in dependency chains.

This is illustrated below:



In this example, if the `Party.Consumer` object is not created (i.e. if the condition PartyType='Consumer' is false and the object is marked as 'do not insert'), then the dependent object `RepeatingAttribute.Address` will not be created either because it has a control attribute that would have a value of NULL.

4.2.2 Modifying a Portrait Foundation object

Once an object has been created it's name, category and type cannot be modified.

The Condition, Mutually exclusive group name and attribute properties may be modified.

To modify a Portrait Foundation object - select the object in the list box and either:

- Click on the 'Properties' button,
- Select 'Properties…' from the context menu, or
- Double click on the object.

The same dialog as shown above will be displayed but with the Name, Category and Type properties set to read-only.

In order to change the Name, Category or Type of an object, the existing object must be deleted and recreated. This prevents any invalid mappings from being created.

4.2.3 Deleting a Portrait Foundation object

To delete a Portrait Foundation object, select it in the list and either:

- Click on the 'Delete' button,
- Select 'Delete' from the context menu, or
- Press the 'Delete' button on the keyboard.

You will be asked to confirm the delete before the object is deleted. Select 'Yes' to continue with the deletion.

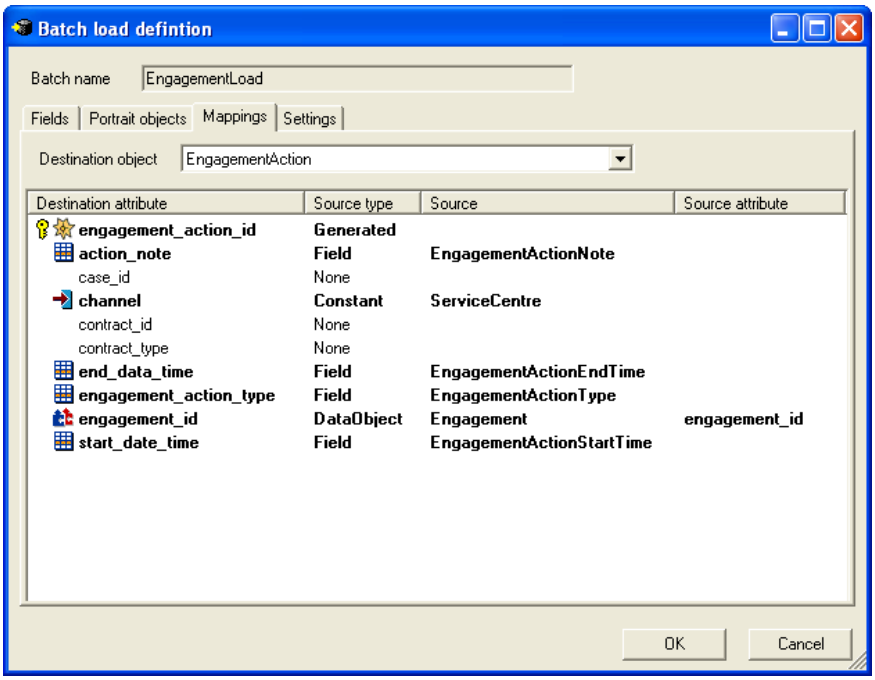
4.2.4 Ordering Portrait Foundation objects

The order of the objects in the list is the order that the objects will be processed for each source data row at runtime. Therefore, any objects that are dependent on primary key assignments from other objects must appear after them in the load order.

The order can be changed by moving them up and down using the  (up) and  (down) buttons.

4.3 Mappings

The mappings tab shows where each of the Portrait Foundation objects that are defined in the load have their attributes populated from and is where mappings are created and maintained.



The **Destination object** combo box can be used to select which object's mappings to display.

To see all mappings for all objects select the '(All objects)' entry.

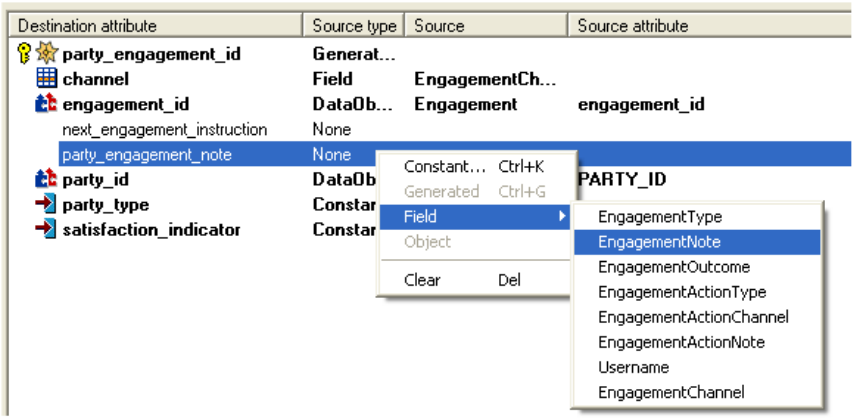
The columns in the list show for each mapping :

- The name of the destination attribute
- The Source type – Generated , Field , Attribute or Constant .
- The Source (depending on the source type)
- If applicable, the source attribute of another object.

The primary key for the object is indicated with the icon.

4.3.1 Adding mappings

Mappings are made by selecting the attribute in the list and selecting the desired source from the context menu.



There are four possible source types for populating the attributes of Portrait Foundation objects:

1 **Field** – The values is source from one of the fields in the staging table. The list of available fields is filtered by the data type of the selected attribute.

2 **Portrait object attribute** – The value is sourced from another object in the load. This option is only available for assigning primary key attributes to corresponding foreign key attributes. For example, mapping the `Party id` of a Party object to the `Party id` attribute on a repeating attribute in order to associate the objects.

3 **Constant** – A constant value. (See Constant values below)

4 **Generated** – The value should be a new generated id. This option is only available on primary key attributes and is assigned automatically.

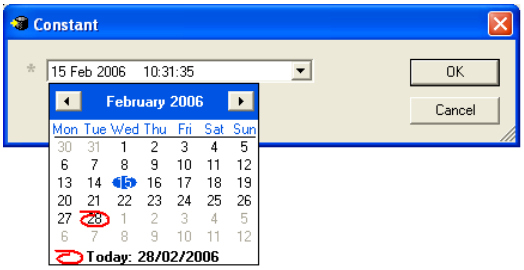
Constant values

The constant values can be set using the 'Constant…' option. A constant dialog for the appropriate data type is displayed.

These dialogs are also used when defining default values for Portrait Foundation object attributes on the 'Portrait object properties' dialog.

Boolean

Date time



Reference data

String and Integer

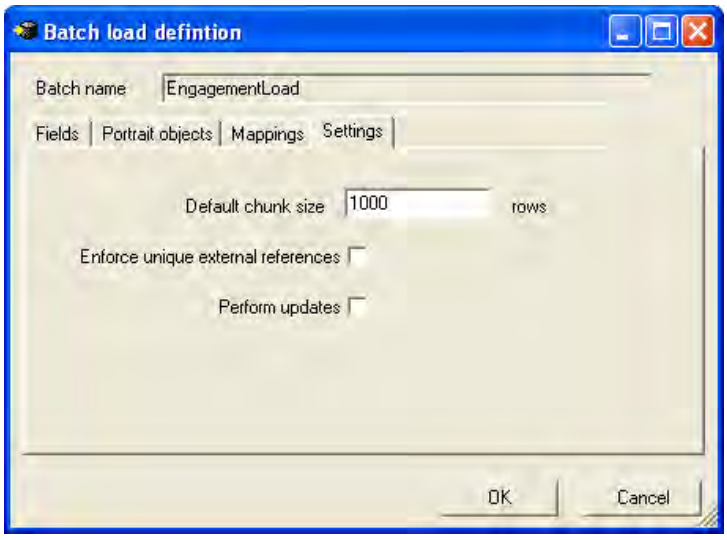
4.3.2 Modifying mappings

Mappings can be modified by simply selecting another value from the context menu.

4.3.3 Deleting a Mapping

To clear a mapping select 'Clear' from the context menu or use the shortcut key 'Delete'.

4.4 Settings



The settings tab allows the following values to be set:

4.4.1 Default chunk size

At runtime the staging table is processed in 'chunks'. A chunk is a set of rows that are bulk loaded within the scope of a transaction. If there is an error when loading the chunk the chunk size is reduced to 10% of the current chunk size and reprocessed until the row that contains the error is identified (i.e. when chunk size = 1).

The chunk size should be set to a level that allows the best balance of performance versus expected error frequency.

The default chunk size is 1000 rows. A chunk size of around 5,000 to 10,000 has been shown to provide the best performance. Obviously, the performance will depend upon the hardware and demand on the database server at load time.

The default chunk size can be modified without using the configuration suite Batch load editor by calling the `p_amc_bl_upd_batch_settings` BLF admin stored procedure.

For more information on setting the default chunk size refer to the document '**Batch Load Framework – Performance statistics and guidelines**'.

4.4.2 Enforce unique external references

The batch load framework allows an external reference string to be provided when initiating a batch load to provide a user defined reference for the batch instance and to help with monitoring and reporting.

By default the external reference does not have to be unique – i.e. the column in the batch instance table does not have a unique constraint applied.

If this option is selected the batch load runtime will check that the value is unique before executing the batch. If an batch instance with the given external reference already exists the batch load runtime will report an error and terminate.

By default this option is not selected.

4.4.3 Perform Updates

The batch load framework supports updating objects if they already exist.

Objects are identified using 'use for existence checking' attributes defined on Portrait Foundation objects (see Portrait Foundation Object attributes).

If this option is selected a set of procedures for each object is created for performing updates.

By default this option is not selected and only inserts are performed.

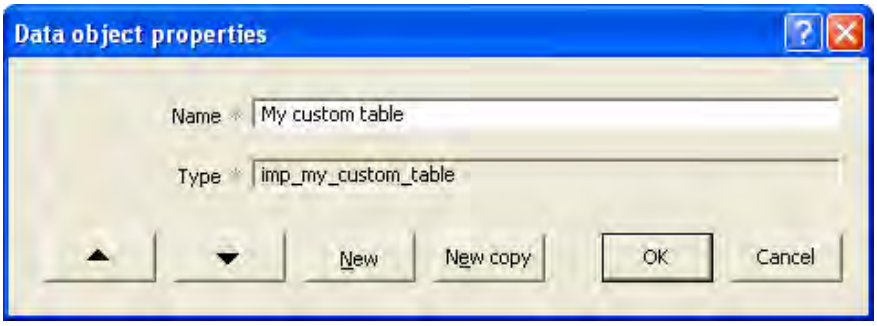
4.5 Custom objects

The batch load framework supports loading into implementation specific tables.

To do this a data object must be created to represent the table where :

- The category must be 'BatchLoading'
- The Type must be the name of the table.
- Each property represents a column in the custom table (subject to Reference data support described below)

For example, if there was a custom table called `imp_my_custom_table`, the data object would be created under the 'BatchLoading' data object category in the Data Objects section of the workspace, as follows:



4.6 Data object properties

Properties are added to the data object in the normal manner.

The type of the property should be selected to match, as close as possible, the physical data type defined on the table.

Data object properties are defined using standard Portrait Foundation data types so the exact data type of the column may not be available.

For simple, non-reference data properties, use the following guidelines for selecting the appropriate data type:

SQLServer type	Portrait data type
• char • nchar • ntext • nvarchar • text • uniqueidentifier • varchar	String
• decimal • float • money • real • smallmoney	Floating point number

SQLServer type	Portrait data type
• bigint • int • smallint • tinyint	Integer
• datetime • smalldatetime • timestamp	Datetime
• binary • bit • cursor • image • varbinary	Not supported

4.6.1 Reference properties

Portrait Foundation provides a set of generic tables for defining reference data. Reference data is used to define a discrete set of values for a named group.

For example, there might be a Titles group that defines the values Mr, Mrs, Miss etc.

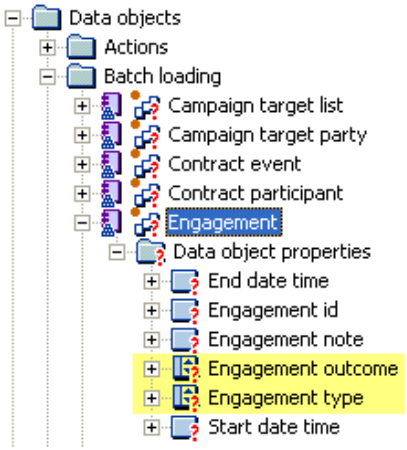
Tables that hold reference data item values do so by defining two columns, one for the reference data group id and one for the reference data item id.

In order to populate columns of this type the value of the ids needs to known at runtime. These values are, in the main, generated at deployment time and the exact values can differ between implementations.

For this reason the source data will provide the **system name** of the item (and sometimes the group as well) and not the id.

The batch load framework supports the resolution of reference data item system names to ids through the use of **reference properties** defined on the data object for the target entity / table.

The data object that defines this entity defines just one Reference property for each reference data value.



The property descriptor for the **physical column** name (see Property descriptors below) must use the column that maps to the `reference_data_item_id`. In the example above this would be `engagement_type_rdi` for the engagement type and `engagement_outcome_rdi` for the outcome.

If there is a source field that provides the system name (string) of the reference data item, this can be mapped to the reference property.

At runtime, the pre-processing stage will resolved all reference data group and items ids and store them in utility columns in the staging table.

The columns on the target table are populated from these utility columns at load time.

Reference properties can be used for populating these pairs of columns on custom tables under the following circumstances:

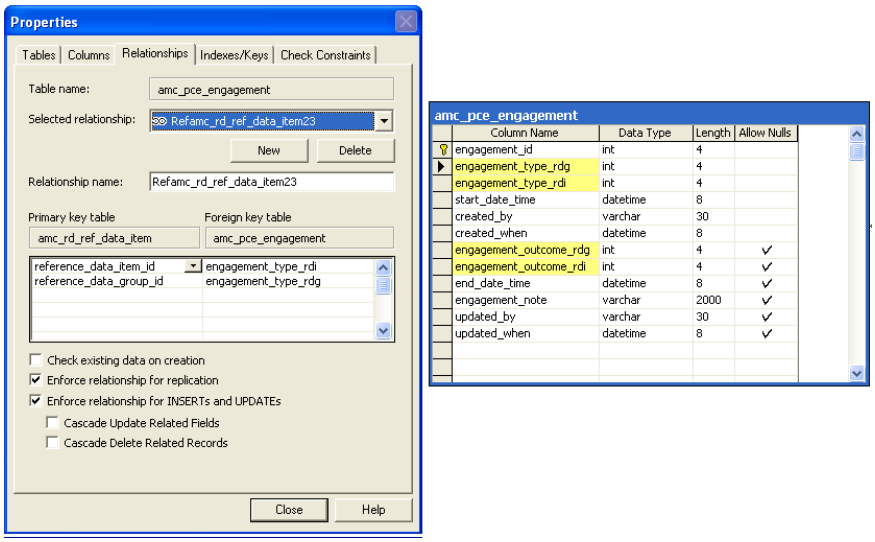
1. Define the columns on the table with a **foreign key constraint** on to `amc_rd_ref_data_item`, or
2. Define columns on the table using a fixed **naming convention**

Foreign key constraint

All Portrait Foundation tables that hold reference data values have foreign key constraints created for them. Constraints can be added to custom / implementation tables in a similar way.

For example the `amc_pce_engagement` table has two foreign key constraints for `engagement_type` and `engagement_outcome`.

These are highlighted below:



If the constraint method is used then the columns can use any naming convention.

The columns that are used in the constraint are looked up in SQL Server system tables at deploy time.

Naming convention

If the implementation chooses not to define foreign key constraints for the reference value columns then the batch load framework can still work out the column names if the following naming convention is used:

- `<column_prefix>_rdi` – for the reference data item column
- `<column_prefix>_rdg` – for the reference data group column

Where `<column_prefix>` is the same for both columns but can be any value so long as it does not contain the substring '`_rdi`'. The batch load framework calculates the name of the `_rdg` column from the `_rdi` column by string substitution.

4.6.2 Property descriptors

In order to provide more information about an attribute of a data object a number of property descriptors have been added.

A property descriptor is used to provide extra metadata about each attribute.

Property descriptors supported include:

- Attribute type
- Physical column
- Primary key
- Foreign key
- Unique
- Mandatory

Attribute type (Mandatory)

This is used to indicate whether the attribute is a fixed column on a table or a configurable column. All properties of custom table data objects should have an Attribute type property descriptor defined with a value of 'FIXED'. If this property descriptor is not specified the property will be ignored by the Batch Load editor.

Physical Column (Mandatory)

The physical column property descriptor is used to store the physical column name that the property represents on the table. All properties must have this property descriptor.

Primary key (Optional)

The Primary key property descriptors is used to indicate which attribute is a primary key. This information is used when generating the staging table so that unique ids can be allocated to new objects. Only primary key attributes can be assigned to other objects attributes that are foreign keys.

Convention is to assign the value 'TRUE', but value can be provided for this descriptor as it is the presence of the descriptor that is relevant not it's value.

Foreign key (Optional)

The foreign key property descriptor is used to indicate that the column is a foreign key to another object. Only foreign key attributes can have primary keys of other objects assigned to them. It is assumed that the Primary key and Foreign key attributes are of the same type at runtime. This allows the primary and foreign keys to be either strings or integer types. It also permits ids defined on configurable objects such as Parties to have their Primary key, which is defined as a string, to be assigned to foreign key attributes that are integers (e.g. party id on engagement action party). The id is only defined as a string on configurable objects because under normal runtime scenarios that integer id, the real underlying type, is carried as an encrypted string.

Convention is to assign the value 'TRUE', but value can be provided for this descriptor as it is the presence of the descriptor that is relevant not it's value.

Unique (Optional)

This property descriptor should be applied to any attributes that are unique. This allows them to be used in existence checking at load run time.

Convention is to assign the value 'TRUE', but value can be provided for this descriptor as it is the presence of the descriptor that is relevant not it's value.

Mandatory (Optional)

The mandatory descriptor is used to indicate which properties are mandatory. This property is not currently used. It was intended to be used to force mappings to be made. This is not currently implemented as a validation check.

Convention is to assign the value 'TRUE', but value can be provided for this descriptor as it is the presence of the descriptor that is relevant not it's value.

The data objects for configurable Portrait Foundation entities, such as Parties and Contracts, have their property descriptors added automatically.

5 Portrait Foundation objects

5.1 Portrait Foundation object tables

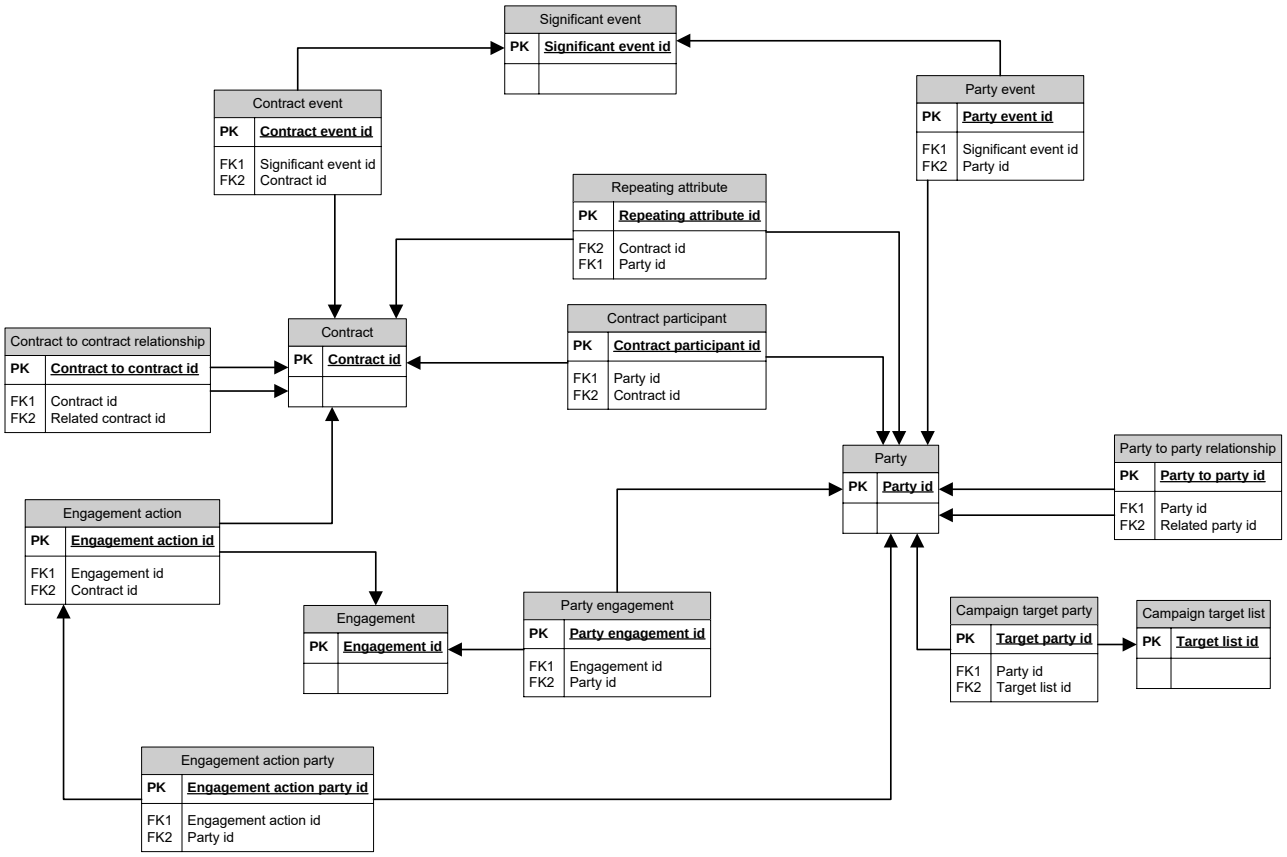
Table 1 – Configurable entities

Portrait Object Category	Table	Notes
Party	amc_pce_party amc_pce_party_type_data amc_pce_party_search amc_pce_party_unique	Creates multiple records in the `amc_pce_party_type_data` table for hierarchical objects. Populating `amc_pce_prty_typ_encr_data` where security credentials are stored is not supported.
Product	amc_pce_contract amc_pce_contract_data amc_pce_contract_search amc_pce_contract_unique	Creates multiple records in the `amc_pce_contract_data` table for hierarchical objects.
Engagement action	amc_pce_engagement_action amc_pce_eng_act_type_data amc_pce_eng_act_search amc_pce_eng_act_unique	Not hierarchical.
Significant event	amc_pce_event amc_pce_event_type_data amc_pce_event_search amc_pce_event_unique	Not hierarchical.
Repeating attribute	amc_pce_repeating_atrb amc_pce_rptng_atrb_search amc_pce_rptng_atrb_unique	Not hierarchical.
Contract to contract relationship	amc_pce_repeating_atrb amc_pce_rptng_atrb_search amc_pce_rptng_atrb_unique	Not hierarchical.
Party to party relationship	amc_pce_repeating_atrb amc_pce_rptng_atrb_search amc_pce_rptng_atrb_unique	Not hierarchical.

Batch loading category type	Table	Notes
Repeating attribute	amc_pce_repeating_atrb	Used to create stub repeating attribute records that could be of any type. The type would typically be provided through a mapping from a source field. No configurable columns can be populated.
Engagement action	amc_pce_engagement_action	Used to create stub engagement action records that could be of any type. The type would typically be provided through a mapping from a source field. No configurable columns can be populated.
Significant event	amc_pce_event	Used to create stub significant event records that could be of any type. The type would typically be provided through a mapping from a source field. No configurable columns can be populated.

Batch loading category type	Table	Notes
Party event	amc_pce_party_event	To associate a party with a particular significant event.
Contract event	amc_pce_contract_event	To associate a contract with a particular significant event.
Contract participant	amc_pce_party_contract	To associate a party with a contract.
Engagement	amc_pce_engagement	All engagement related objects (engagement actions, party engagements etc.) require an engagement object to be created first.
Engagement action party	amc_pce_engmt_action_party	For relating a party to an engagement action.
Party engagement	amc_pce_party_engagement	For relating a party to an engagement.

5.2 Portrait Foundation object relationships



6 Glossary

System name

System names are used to name items within configuration. In the context of the Batch Load Framework they can be are used to generate the names for database identifiers such as stored procedure and table names.

The system names used in the batch load framework must conform to the following rules:

1. The first character must be :

- A letter as defined by the Unicode Standard 2.0. The Unicode definition of letters includes Latin characters from a through z and from A through Z, in addition to letter characters from other languages.
- The underscore (_) character.

2. Subsequent characters can be:

- Letters as defined in the Unicode Standard 2.0.
- Decimal numbers from either Basic Latin or other national scripts.
- The underscore character.

Embedded spaces or special characters are not allowed.

Staging table

A generated table in the operational database where values for the load are sourced from at runtime.

Batch definition

The set of Fields, Portrait Foundation objects and Mappings defined in the configuration suite that are used to perform the load at runtime.

Portrait object

A logical view of an entity with a Portrait Foundation system as opposed to the underlying database structure where they are stored. Portrait Foundation objects are used in Processing modelling and other configuration activities.