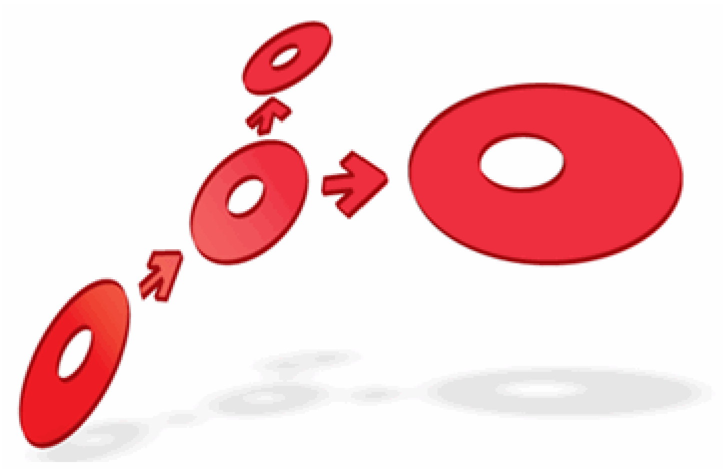
Portrait
Foundation



# Database Views User Guide

Edition 1.3

10 January 2013

Pitney Bowes
Software

# Foundation
# Database Views User Guide

©2013
**Copyright Portrait Software International Limited**

All rights reserved. This document may contain confidential and proprietary information belonging to Portrait Software plc and/or its subsidiaries and associated companies.

Portrait Software, the Portrait Software logo, Portrait, Portrait Software's Portrait brand and Million Handshakes are the trademarks of Portrait Software International Limited and may not be used or exploited in any way without the prior express written authorization of Portrait Software International Limited.

## Acknowledgement of trademarks

Other product names, company names, marks, logos and symbols referenced herein may be the trademarks or registered trademarks of their registered owners.

## 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 describes how to use the Database view feature introduced in Portrait Foundation 4.2 that was initially added for the purpose of data paging.

## Intended audience

Developers and configurers who need to access CBE data in a normalised state from the Portrait Foundation operational database rather than using the Datamart.

## Software release

Portrait Foundation 4.4 or later.

View Builder 2.0 was released as part of 4.2 Service Pack 2.

# Contents

# 1 Introduction

Portrait Foundation version 4.2 introduced a new capability to generate database views representing individual configured business entities or combinations of entities on the Foundation operational database. These views are easier to work with than the underlying database tables which are designed to support a completely flexible logical data model and hence are not easy to query directly.

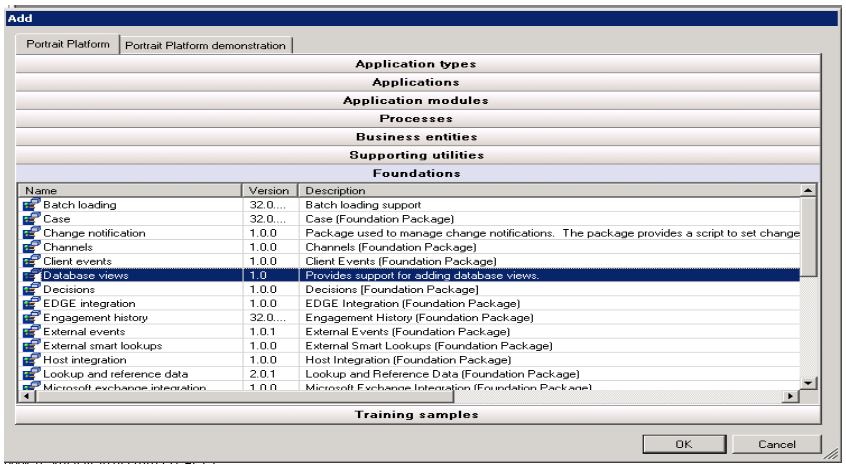
View builder is the application that has been developed to generate these Portrait Foundation operational database views. It is integrated into the Configuration Suite, but can also be run as a separate standalone application.

# 2 View Builder 2.0

## 2.1 Configuration Suite integration

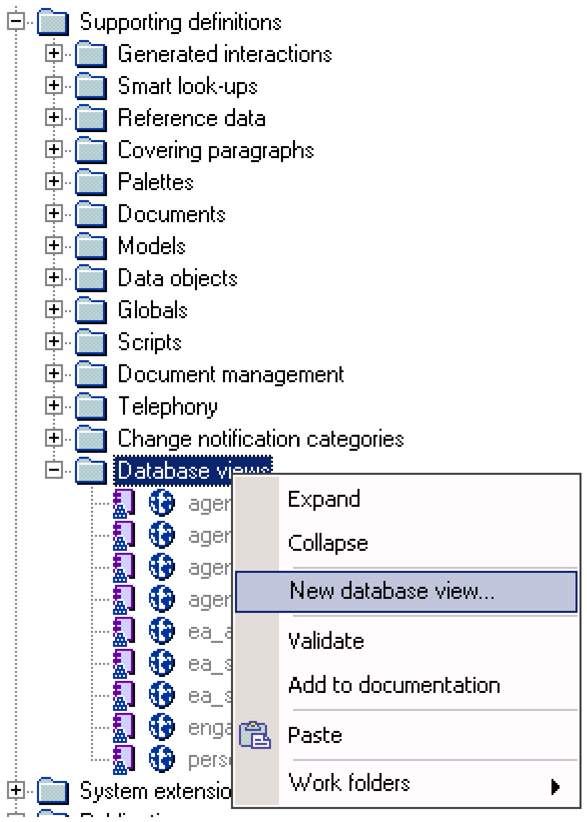
Inside the Configuration Suite, View builder is the Database view definition editor.

To be able to create and edit database views in the Configuration Suite, you must first include the "Database views" package into your package or workspace. This can be found on the Portrait Platform tab under Foundations. Once added, you will find a new node in the tree called "Database views," under the "Supporting definitions" node.

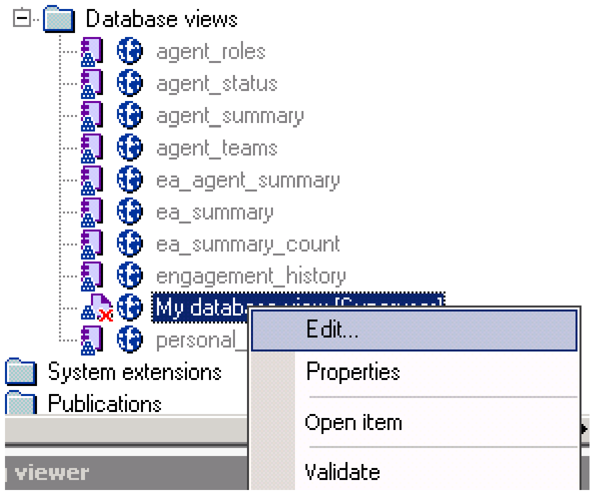


| Name | Version | Description |
|---|---|---|
| Batch loading | 32.0.... | Batch loading support |
| Case | 32.0.... | Case (Foundation Package) |
| Change notification | 1.0.0 | Package used to manage change notifications. The package provides a script to set change |
| Channels | 1.0.0 | Channels (Foundation Package) |
| Client events | 1.0.0 | Client Events (Foundation Package) |
| Database views | 1.0 | Provides support for adding database views. |
| Decisions | 1.0.0 | Decisions (Foundation Package) |
| EDGE integration | 1.0.0 | EDGE Integration (Foundation Package) |
| Engagement history | 32.0.... | Engagement History (Foundation Package) |
| External events | 1.0.1 | External Events (Foundation Package) |
| External smart lookups | 1.0.0 | External Smart Lookups (Foundation Package) |
| Host integration | 1.0.0 | Host Integration (Foundation Package) |
| Lookup and reference data | 2.0.1 | Lookup and Reference Data (Foundation Package) |
| Microsoft exchange integration | 1.0.0 | Microsoft Exchange Integration (Foundation Package) |

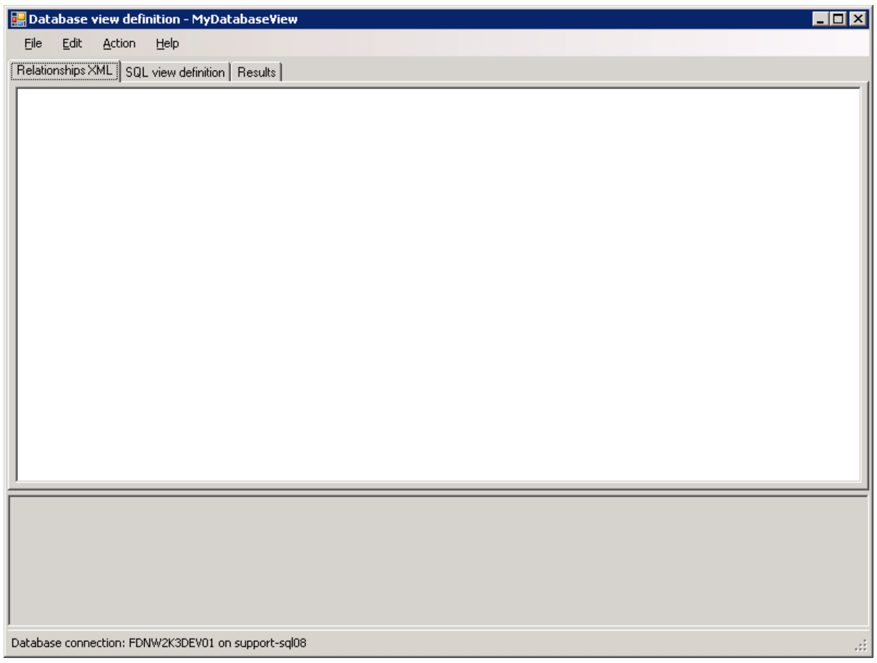
Navigate to the "Supporting definitions" folder in your package or workspace. Select "New database view..." from the context menu of the "Database views" node. Enter a display name and system name for the database view that you wish to create.



To open the database view definition editor right click on your newly created database view and select "Edit…".



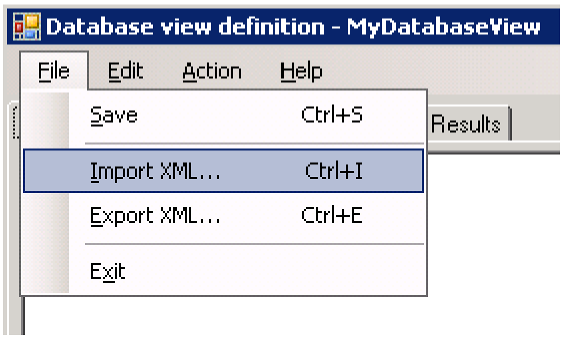
This displays the Database view definition editor.

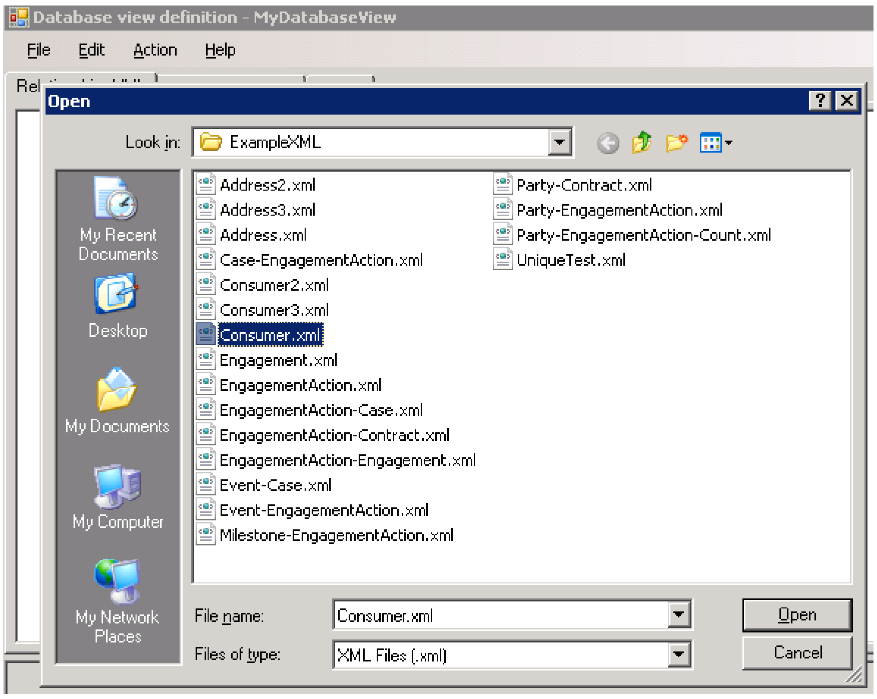


Database connection: FDNW2K3DEV01 on support-sql08

The XML definition of your view can be entered directly in this dialog or imported from file. Section 3 contains details of the view definition XML schema and examples of how to retrieve and join different CBE data.

To load a file, select "Import XML…" from the "File" menu and select the relevant XML file.



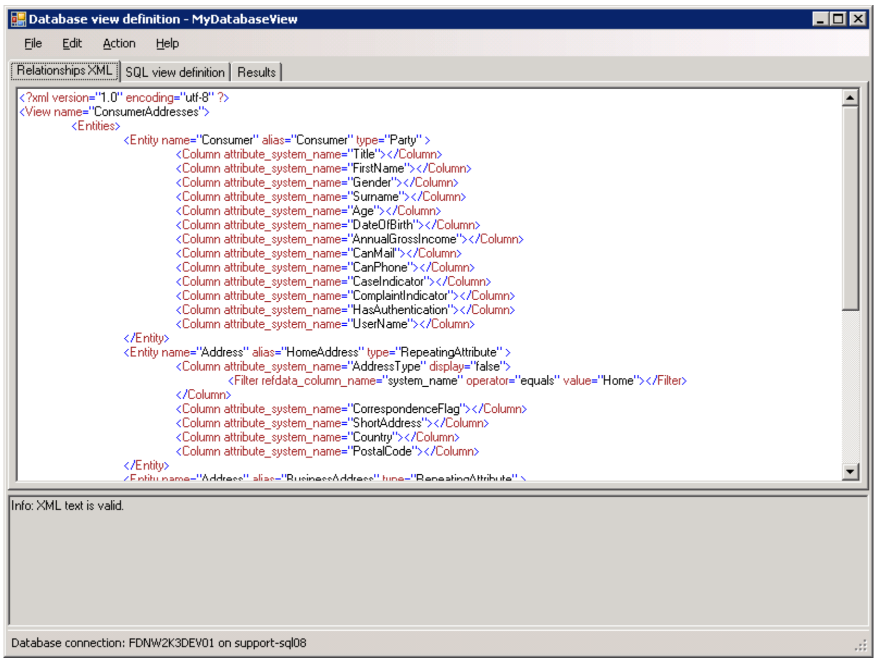


Look in: ExampleXML

Address2.xml
Address3.xml
Address.xml
Case-EngagementAction.xml
Consumer2.xml
Consumer3.xml
Consumer.xml
Engagement.xml
EngagementAction.xml
EngagementAction-Case.xml
EngagementAction-Contract.xml
EngagementAction-Engagement.xml
Event-Case.xml
Event-EngagementAction.xml
Milestone-EngagementAction.xml
Party-Contract.xml
Party-EngagementAction.xml
Party-EngagementAction-Count.xml
UniqueTest.xml

File name: Consumer.xml
Files of type: XML Files (.xml)

Once the file is loaded, it will automatically be validated against the XML schema definition. All results are displayed in the information view below the tabs.

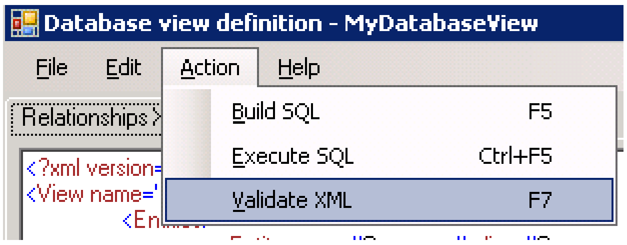


```
<?xml version="1.0" encoding="utf-8" ?>
<View name="ConsumerAddresses">
        <Entities>
                <Entity name="Consumer" alias="Consumer" type="Party" >
                        <Column attribute_system_name="Title"></Column>
                        <Column attribute_system_name="FirstName"></Column>
                        <Column attribute_system_name="Gender"></Column>
                        <Column attribute_system_name="Surname"></Column>
                        <Column attribute_system_name="Age"></Column>
                        <Column attribute_system_name="DateOfBirth"></Column>
                        <Column attribute_system_name="AnnualGrossIncome"></Column>
                        <Column attribute_system_name="CanMail"></Column>
                        <Column attribute_system_name="CanPhone"></Column>
                        <Column attribute_system_name="CaseIndicator"></Column>
                        <Column attribute_system_name="ComplaintIndicator"></Column>
                        <Column attribute_system_name="HasAuthentication"></Column>
                        <Column attribute_system_name="UserName"></Column>
                </Entity>
                <Entity name="Address" alias="HomeAddress" type="RepeatingAttribute" >
                        <Column attribute_system_name="AddressType" display="false">
                                <Filter refdata_column_name="system_name" operator="equals" value="Home"></Filter>
                        </Column>
                        <Column attribute_system_name="CorrespondenceFlag"></Column>
                        <Column attribute_system_name="ShortAddress"></Column>
                        <Column attribute_system_name="Country"></Column>
                        <Column attribute_system_name="PostalCode"></Column>
                </Entity>
                <Entity name="Address" alias="BusinessAddress" type="RepeatingAttribute" >
```

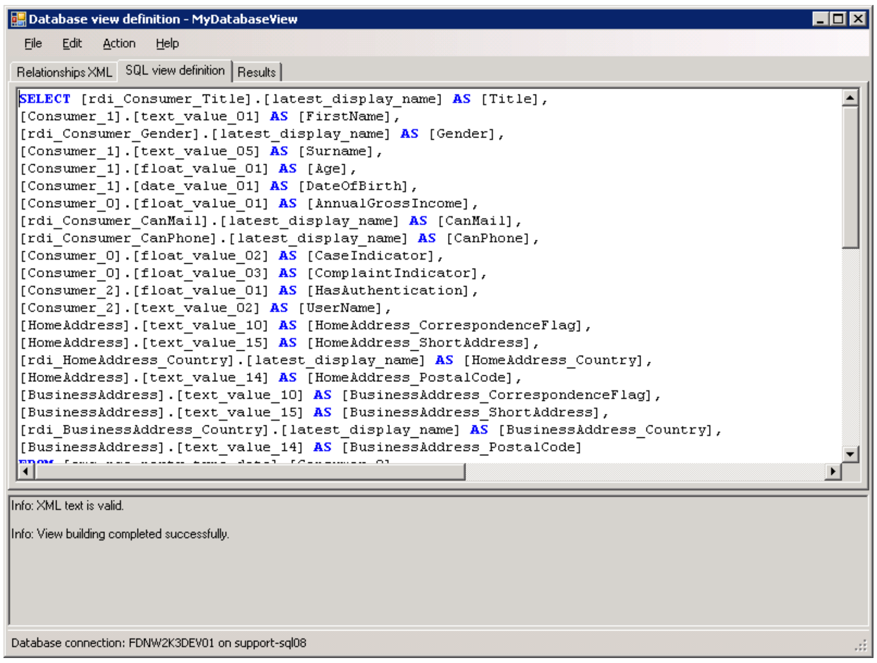
Info: XML text is valid.

Database connection: FDNW2K3DEV01 on support-sql08

The validity of your view definition XML can be tested at any time by selecting “Validate XML” from the “Action” menu or by pressing F7.



This validation step ensures that the XML contains valid elements, but it does not guarantee that it will generate the correct view definition. To test the view definition itself, select “Build SQL” from the “Action” menu or press F5. The “SQL view definition" tab should now display the SQL that was built from the XML definition, as shown below. Any errors are reported in the information view below the tabs. The XML definition needs to be fixed before continuing.



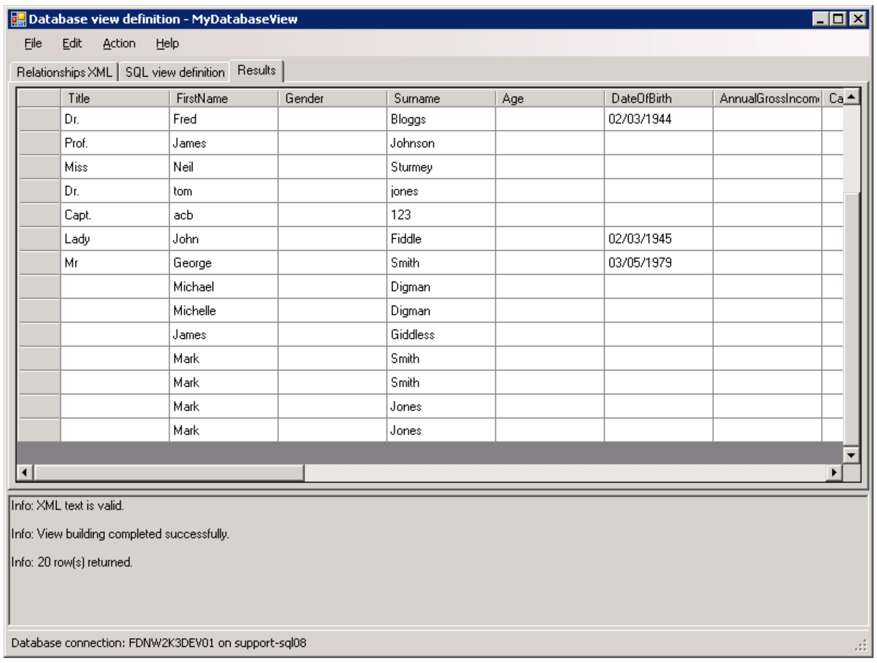
```
SELECT [rdi_Consumer_Title].[latest_display_name] AS [Title],
[Consumer_1].[text_value_01] AS [FirstName],
[rdi_Consumer_Gender].[latest_display_name] AS [Gender],
[Consumer_1].[text_value_05] AS [Surname],
[Consumer_1].[float_value_01] AS [Age],
[Consumer_1].[date_value_01] AS [DateOfBirth],
[Consumer_0].[float_value_01] AS [AnnualGrossIncome],
[rdi_Consumer_CanMail].[latest_display_name] AS [CanMail],
[rdi_Consumer_CanPhone].[latest_display_name] AS [CanPhone],
[Consumer_0].[float_value_02] AS [CaseIndicator],
[Consumer_0].[float_value_03] AS [ComplaintIndicator],
[Consumer_2].[float_value_01] AS [HasAuthentication],
[Consumer_2].[text_value_02] AS [UserName],
[HomeAddress].[text_value_10] AS [HomeAddress_CorrespondenceFlag],
[HomeAddress].[text_value_15] AS [HomeAddress_ShortAddress],
[rdi_HomeAddress_Country].[latest_display_name] AS [HomeAddress_Country],
[HomeAddress].[text_value_14] AS [HomeAddress_PostalCode],
[BusinessAddress].[text_value_10] AS [BusinessAddress_CorrespondenceFlag],
[BusinessAddress].[text_value_15] AS [BusinessAddress_ShortAddress],
[rdi_BusinessAddress_Country].[latest_display_name] AS [BusinessAddress_Country],
[BusinessAddress].[text_value_14] AS [BusinessAddress_PostalCode]
```

```
Info: XML text is valid.

Info: View building completed successfully.
```

Database connection: FDNW2K3DEV01 on support-sql08

Once the SQL has been generated, sample rows can be viewed by selecting "Execute SQL" from the "Action" menu or by pressing Ctrl+F5.



| Title | FirstName | Gender | Surname | Age | DateOfBirth | AnnualGrossIncom | Ca |
|---|---|---|---|---|---|---|---|
| Dr. | Fred | | Bloggs | | 02/03/1944 | | |
| Prof. | James | | Johnson | | | | |
| Miss | Neil | | Sturmey | | | | |
| Dr. | tom | | jones | | | | |
| Capt. | acb | | 123 | | | | |
| Lady | John | | Fiddle | | 02/03/1945 | | |
| Mr | George | | Smith | | 03/05/1979 | | |
| | Michael | | Digman | | | | |
| | Michelle | | Digman | | | | |
| | James | | Giddless | | | | |
| | Mark | | Smith | | | | |
| | Mark | | Smith | | | | |
| | Mark | | Jones | | | | |
| | Mark | | Jones | | | | |

```
Info: XML text is valid.

Info: View building completed successfully.

Info: 20 row(s) returned.
```

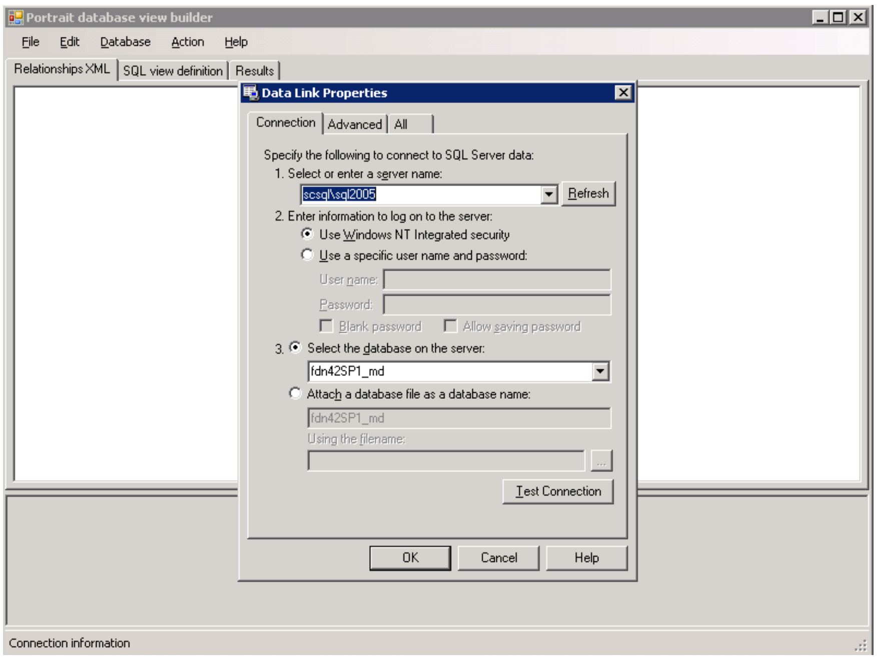
Database connection: FDNW2K3DEV01 on support-sql08

When you are satisfied with the results returned from the view, select "Save" from the "File" menu. This will close the editor and save the view definition XML in the "Database view" node. If the XML is not valid, you will be unable to save it. To create this view in your operational database, open the deployer and deploy your latest configuration.

## 2.2 Standalone application

View builder can also be run as a standalone application and is designed to be used as a developer tool. It can be launched by running **ViewBuilder.exe** from <*Portrait_Foundation_Install_Folder*>\Common\Bin (e.g. C:\Program Files\PST\Portrait Foundation\Common\Bin). For projects using Portrait Foundation 4.2 and above, it is recommended to use the Configuration Suite integration as it manages the deployment of views across the project in a consistent way.

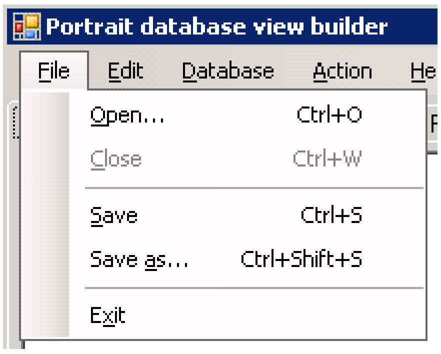
When launched as a standalone application, you must first select your Portrait Foundation operational database.



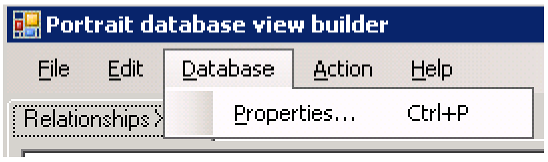
Specify the following to connect to SQL Server data:

1. Select or enter a server name: scsql\sql2005
2. Enter information to log on to the server:
   - [✓ Use Windows NT Integrated security]
   - Use a specific user name and password:
   - User name:
   - Password:
   - Blank password
   - Allow saving password
3. [✓ Select the database on the server:] fdn42SP1_md
   - Attach a database file as a database name: fdn42SP1_md
   - Using the filename:

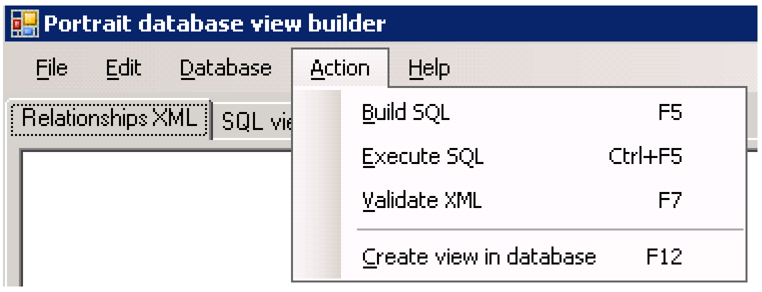
The menu items are a little different in standalone mode. You can manage your XML definition files via the "Open…", "Close", "Save" and "Save as…" options on the "File" menu.



You can select a different Portrait Foundation operational database via the "Database" menu.



Finally you can choose to create the database view directly on your Portrait Foundation operational database by selecting "Create view in database" from the "Action" menu or by pressing F12.


Portrait database view builder
File
Edit
Database
Action
Help
Relationships XML
SQL vie
Build SQL F5
Execute SQL Ctrl+F5
Validate XML F7
Create view in database F12

# 3 View definition XML

## 3.1 Schema

A view definition consists of a set of entities and a set of relationships between those entities.

### Entity

The entity represents a logical entity within the configured Portrait Foundation system, such as a Consumer or a Mortgage product.

The following attributes apply to the Entity element.

- **name** Mandatory. The system name of the entity, as defined in Portrait Foundation Configuration (e.g. Address).
- **type** Mandatory. Type is the CBE category, for those supported entities which are CBEs. For example, name="Consumer" type="Party". The following CBE categories are valid Entity types.
    - Party
    - Contract
    - RepeatingAttribute
    - Case
    - SignificantEvent
    - Milestone
    - EngagementAction

  In addition, the following are also valid types.

    - ReferenceData
    - TaskDefinition
    - Other (see below for description)

  For the ReferenceData type, the name is the system name of the reference data group.

  For the TaskDefinition type, the name is the system name of the workflow task.

  The value "Other" can be used to represent the following data object entities.

    - Engagement action (Engagement, Action). To retrieve data for all types of engagement action, set the name attribute to "EngagementActionType".
    - Engagement history (Engagement, History). To retrieve engagement history data, set the name attribute to "Engagement".
    - Engagement involved party (Engagement, InvolvedParty). To retrieve the parties linked to an engagement action, set the name attribute to "EngagementActionParty".
    - Engagement party (Engagement, Party). To retrieve the parties linked to an engagement, set the name attribute to "PartyEngagement".
    - Contract participant (ContractParticipants, ContractParticipant). To retrieve the parties linked to a contract, set the name attribute to "PartyContract".

  - Task history (Task, History). To retrieve the history linked to a workflow task, set the name attribute to "TaskHistory".

  To access the data object properties for "Other" entity types use the system_column_name. See "Other entities..." on the Help menu for the available data object properties.

- **alias** Mandatory. The name by which the entity is known within the scope of the view (e.g. HomeAddress).

One entity can be declared multiple times if a different alias is used each time, for example HomeAddress and BusinessAddress (see Consumer example).

**NB:** Automatic tasks generated by the Create Auto Task node are not supported as there is nothing to query.

## Column element

An Entity can contain a set of Column elements, which represent the columns to be retrieved in the view. A column can be sourced from one of two places – a configured "attribute" of the entity (e.g. Postcode) or a fixed "system" column (e.g. party_id, contract_id, engagement_id, etc).

Attribute columns can represent either a simple type (string, date, etc) or a complex type based on reference data. For reference data types (e.g. AddressType), the display name is returned by default. It is possible to override this behaviour and return the system name or id of the reference data item instead, by setting the refdata_column_name attribute.

The following attributes can be applied to the Column element:

- **attribute_system_name** The system name of an attribute defined for the Entity. It is recommended to use this rather then attribute_display_name.
- **attribute_display_name** The display name of the attribute defined for the Entity. Declare either a system name or a display name, not both. Be aware that the Display Name is not unique.
- **system_column_name** The name of a column on the underlying physical table (e.g. party_id, created_when, effective_to, effective_from, etc).
- **type** Optional (can usually be inferred from other attributes). If provided, it must be "attribute", "system" or "calculated". Specify calculated if you wish to use a function (e.g. Count or DateDiff).
- **refdata_column_name** Optional. This is used with a reference data attribute to specify the name of the column to retrieve from the reference data item table (e.g. system_name, reference_data_item_id, reference_data_group_id, etc). Default is "latest_display_name".
- **view_column_name** Mandatory if type is "calculated". The name by which the column will appear in the view. It defaults to the Attribute's SystemName for an attribute column or EntityName_ColumnName for a system column. If multiple columns have the same name (either automatically-built names or user-supplied names) they will get prefixed with the name of the entity.
- **display** Optional. If "false" the column will not be included in the view. Default is "true".
- **search** Optional. If "true" the SQL is built to use the searchable attributes table. This only has effect when used with attributes which are configured as searchable or unique. The default is "false".
- **function** Mandatory if type is "calculated". The SQL function to call including its parameters in brackets. The function should contain the whole function definition, with any parameters defined using the C# style composite format string (e.g."DateDiff('mi',{0},{1})"). Fixed string parameters must use single quotes. To use column attributes as parameters see Engagement example. **NB:** For the "TaskDefinition" Entity type only the *COUNT* function is supported.
- **group_by** Optional. Contains an index that represents the position of the column in the GROUP BY clause of the view SQL. If a function like Count is

used, then each selected column would require a different index (see Party-EngagementAction-Count example).

One of attribute_system_name, attribute_display_name or system_column_name must be provided for a column element. It is not valid to have more than one.

For performance reasons it is always a good idea to only select the columns you require.

**NB:** Attributes which are configured as encrypted are not supported and will not be returned.

## Filter element

A Column element can contain one or more Filter elements. A Filter element defines a restriction on a particular column.

The view will only return rows that meet the filter criteria. The filter forms part of the WHERE clause in the view SQL.

An example of filter use is to retrieve only a subset of repeating attributes related to a party. For example, retrieve party addresses where address type = "Home" (see Address example). For reference data column types (e.g. AddressType), the system name is used by default.

The following attributes can be applied to a Filter element:

- **refdata_column_name** Optional. If the column represents a reference data attribute, refdata_column_name specifies which column to use in the comparison (e.g. latest_display_name). Default is "system_name".
- **operator** Optional. The possible values are
    - equals
    - notequals
    - lessthan
    - lessthanequalto
    - morethan
    - morethanequalto
    - isnot
    - is
    - in

  The "is" and "isnot" values are for NULL evaluations. The "in" value allows a comma delimited list of values. String values in this list must be in single quotes. Default is "equals".

- **value** Mandatory. The comparison value for the filter (e.g. "Home"). For date column types, the value should be in UTC format (YYYY/MM/DD). Other formats may not be recognised depending on the local settings for the SQL Server installation. This attribute setting can only contain one value. SQL functions are not supported.

For performance reasons try to only use filters, that have database indexes. These include:

- Columns that are configured as searchable or unique. Use the search setting on the Column entity.
- Reference data columns using the "system_name" column.
- System columns like party_id (primary keys), effective_to and effective_from.

## Parameter element

A Column element can contain one or more Parameter elements. A Parameter element defines a function parameter declared in the function attribute of a calculated Column. There must be one parameter element for each parameters declared. The first parameter element goes in {0}, the second in {1}, the third in {2} and so on. There is no limit to the number of parameters.

A Parameter is sourced in exactly the same way as a Column entity. From a configured "attribute" of the entity or a fixed "system" column.

The following attributes can be applied to a Parameter element:

- **attribute_system_name** The system name of an attribute defined for the Entity.
- **system_column_name** The name of a column on the underlying physical table (e.g. party_id, created_when, effective_to, effective_from, etc).

One of attribute_system_name or system_column_name must be provided for a column element. It is not valid to have more than one.

## Relationship

A Relationship specifies that two Entities are related to each other in some way. Depending on the types of Entities involved, the relationship needs to declare different information.

In its simplest form, a Relationship contains two Entity elements. These Entities must be declared in the Entities section otherwise the Relationship is not valid. In some cases two Entities are related via a third "Link" entity.

The following attributes can be applied to the Relationship element:

- **name** Mandatory for Party-Contract relationships and not applicable to any other type of relationship. The name specifies the system_name of the party to contract relationship type.
- **exclude_null_results** Optional. If true, rows are excluded from the results if either entity has a null value in the column that joins the entities. In SQL terms an INNER JOIN is used if exclude_null_results is set to true, otherwise a LEFT OUTER JOIN is used. Default is "false".

## Related entity

A one to many relationship is defined using exactly two related entity elements. The system infers the direction of the relationship from the entity definitions.

The following attributes can be applied to the RelatedEntity element.

- **alias** Mandatory. Refers to the alias of an Entity in the Entities section.
- **property** Mandatory when the relationship includes an Entity of type ReferenceData. It is Ignored for other types of Entity. This attribute contains the system name of the reference data property on the "source" Entity. The other Entity must be of type ReferenceData.

## Link entity

A many to many relationship is defined using exactly two related entity elements and one link entity element. The link entity is the linking entity in the many to many relationship. A LinkEntity element is similar to a RelatedEntity element and must have an alias attribute defined for it. The entity which it refers to must contain key fields pointing to each of the related entities. Attributes can be declared for the link entity.

## 3.2 Samples

The following XML sample files can be found in the Portrait Foundation install folder under ExampleXML (i.e. C:\Program Files\PST\Portrait\ExampleXML).

| XML file name | Description |
|---|---|
| Address | Address with filter on the address type. |
| Address2 | Address, illustrating the display tag. |
| Address3 | Address, illustrating “morethan” date filter. |
| Consumer | Joins a consumer with the current home address and business address. |
| Consumer2 | Joins a consumer with its correspondence address. |
| Consumer3 | Consumer entity with “Is Null” filter. |
| Engagement | View of all Engagements with function to calculate duration. Uses function column parameters. |
| EngagementAction | View of a specific Engagement Action “NotifyChangeOfAddress” and it’s configured attributes. |
| EngagementAction-Case | One to many relationship between a case and the engagement actions related to it. Using exclude_null_results to only return engagement actions that are linked to cases. |
| EngagementAction-Contract | One to many relationship between LoanBase contracts and the related engagement actions. Using exclude_null_results to only return engagement actions that are linked to the contracts. |
| EngagementAction-Engagement | View of all Engagements and their Engagement actions. See use of view_column_name. |
| EngagementAction-Milestone | One to many relationship between a milestone and the engagement actions related to it. |
| Event-Case | One to many relationship between a case and the significant events related to it. |
| Event-EngagementActions | One to many relationship between an event and the engagement actions related to it. |
| Party-Contract | Many to Many relationship between Party and Contract. Note that the linking table – pce_party_contract – is not an entity and is not stated in the xml. |
| Party-EngagementAction | List all Agents with the “InboundCallCompleted” engagement action. Uses the EngagementActionParty entity as the link entity. |
| Party-EngagementAction-Count | Uses the function and group_by attributes to count the number of different engagement actions an agent has performed. |
| ReferenceData | View of all items in a reference data group. |
| TaskActionCount-Agent | Uses the function and group_by attributes to count the number of different task related actions an agent has performed. |

| | |
|---|---|
| TaskAttributes | View of a specific Task definition (MakeOutboundCall) with its related attributes |
| TaskAttributes-Count | Uses the function and group_by attributes to count related attribute of the specific task definition. |
| TaskAttributes-Filter | View of a subset of attributes of a specific task definition based on the Filter applied. The Filter can be edited to suit custom requirements. |
| Task-ClassifyComplaint-Party | View of a specific TaskDefinition “ClassifyComplaint” with the party and the repeating attributes. Note that the link entity is the Complainant Repeating attribute type. |

Most of these samples can be run (built & deployed) against a deployed All Applications Workspace.