Portrait
Foundation

Generic Access Implementation Guide

Edition 6.0

12 June 2014

Pitney Bowes
Software

Portrait Foundation
Generic Access Implementation 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 gives a summary of capabilities of the Generic Access components and goes on to provide detailed instructions to implementers on how to use those components.

Restrictions and example usage scenarios for the Generic Access components are provided.

Intended audience

Members of project teams that intend to access external systems and applications using OLE DB providers or the SOAP protocol.

Related documents

None

Software release

Portrait Foundation 5.0 or later.

Portrait

Contents

1 Introduction

Portrait Foundation version 2.3 introduced the Host Integration Framework (HIF). The framework provides support for integration to external systems through a toolkit and some sample components. It is intended as an extensible mechanism, with the extensions being provided through development activities.

The aim of the components described here is to introduce new wrappers of functionality around the existing HIF. This provides a consistent environment in which business and technical users can quickly and easily connect Portrait Foundation DATs to external systems without the need for developer input. The most common technologies – databases and Web services – have been addressed first, but the approach may be extended to other technologies in the future.

This document provides:

- A summary of capabilities of the Generic Access components.
- Detailed instructions to implementers on how to use those components.
- Restrictions and example usage scenarios for the first release of the Generic Access components.

1.1 Outline capabilities

Support is provided for accessing systems through:

- Generic Database Access – GDA:
  - provides access to any system for which an OLE DB provider is available
  - built using standard OLE DB interfaces to achieve a wide coverage of systems (vendor-specific capabilities may not be available)
  - supports the execution of SQL statements and stored procedures/functions
  - provides conversions between Portrait Foundation and OLE DB data types that can be selected on a system basis and on a transaction parameter/output basis
- Generic Web Service Access – GWSA:
  - provides access to any system that may be accessed using the SOAP protocol (over HTTP)
  - built using .Net framework classes and therefore only subject to the Microsoft implementation of the SOAP protocol
  - provides fixed conversions between Portrait Foundation and SOAP (.Net) data types (the ability to tune this conversion, including optional plug-in type converters, may be provided in a later release)

2 Common Generic Access components

There are a number of common user interface components for managing each type of external system and its transactions irrespective of the technology involved.

2.1 System connection information

This is captured in the Management Console. Here there is some technology dependency. For example, for a Web service we capture the URL of the WSDL relating to the managed system, whereas for a database we capture the connection string. Each technology has its own properties tab within the Generic Access category in the Management Console.

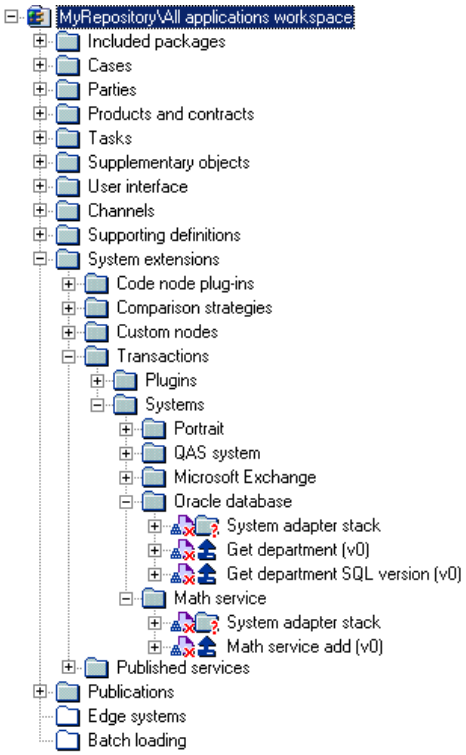
Each system recorded in the Management Console relates to a physical database connection or Web service.

2.2 Configuration tree

Systems using Generic Database Access and Generic Web Service Access components are placed in the same folder as all other Host Integration systems. Each of these systems may be considered as a logical system that is tied to the physical connection (see above). It is possible to configure many logical systems that use the same physical connection, for example, to create groups of related transactions.

Transactions appear below their respective systems as normal.

Figure 1 – Configuration tree placement



2.3 System editor

A standard system editor is used in the Configuration Suite for all systems supported by Generic Access components. Each type of Generic Access component supplies its own technology-specific pane to the editor. The system editor allows you to select one of the external systems defined in the Management Console. The technology-specific pane displays details of the selected system, such as database server name.

2.4 Transaction editor

A standard transaction editor is used in the Configuration Suite for all systems supported by Generic Access components. Each type of Generic Access component supplies its own technology-specific **Command** pane to the editor. The transaction editor allows you to specify a transaction name, system name and description. It also provides tabs containing **Command**, **Inputs** and **Outputs** panes. The **Inputs** and **Outputs** panes are common across all types of system. They allow you to configure all DAT inputs and outputs and to map those inputs and outputs to command inputs and outputs.

The **Command** pane is where the technology-specific configuration is captured. For example, for the Generic Database Access components, this pane captures details of the SQL statement or stored procedure to be executed and details of any parameters, return values and output columns.

2.4.1 Mappings

Command inputs and outputs must be mapped from/to DAT inputs and outputs.

The Generic Access components use the standard Portrait Foundation mapping dialog (used in several places throughout the Configuration Suite, including model usage mappings). This dialog is launched from within the Inputs and Outputs tabs to edit the mappings.

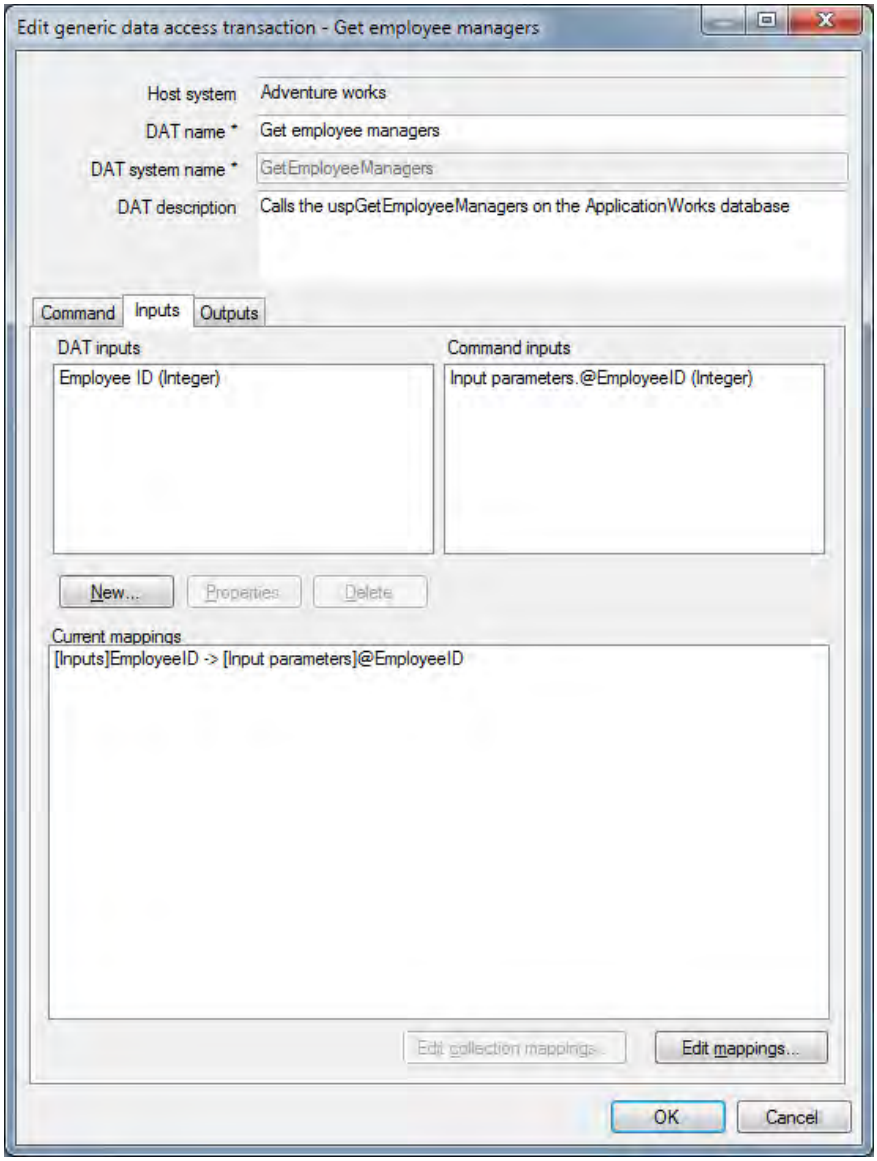
Note that collections may be mapped separately from scalar types (depending on the type of Generic Access component). This gives you the ability to restrict the properties to be mapped and to map between collections of different types. This may be useful when populating a data object collection from a record set which contains columns that can be discarded.

The Portrait Foundation mapping dialog however has some restrictions, such as an inability to index into collections to select only the first element. This may be resolved in a future release.

2.4.1.1 Inputs tab

The Inputs tab displays the mappings from DAT inputs to command inputs (shown with their corresponding Portrait Foundation data types).

Figure 2 – Example Inputs tab



DAT Inputs are managed using the **New…**, **Properties** and **Delete** buttons.

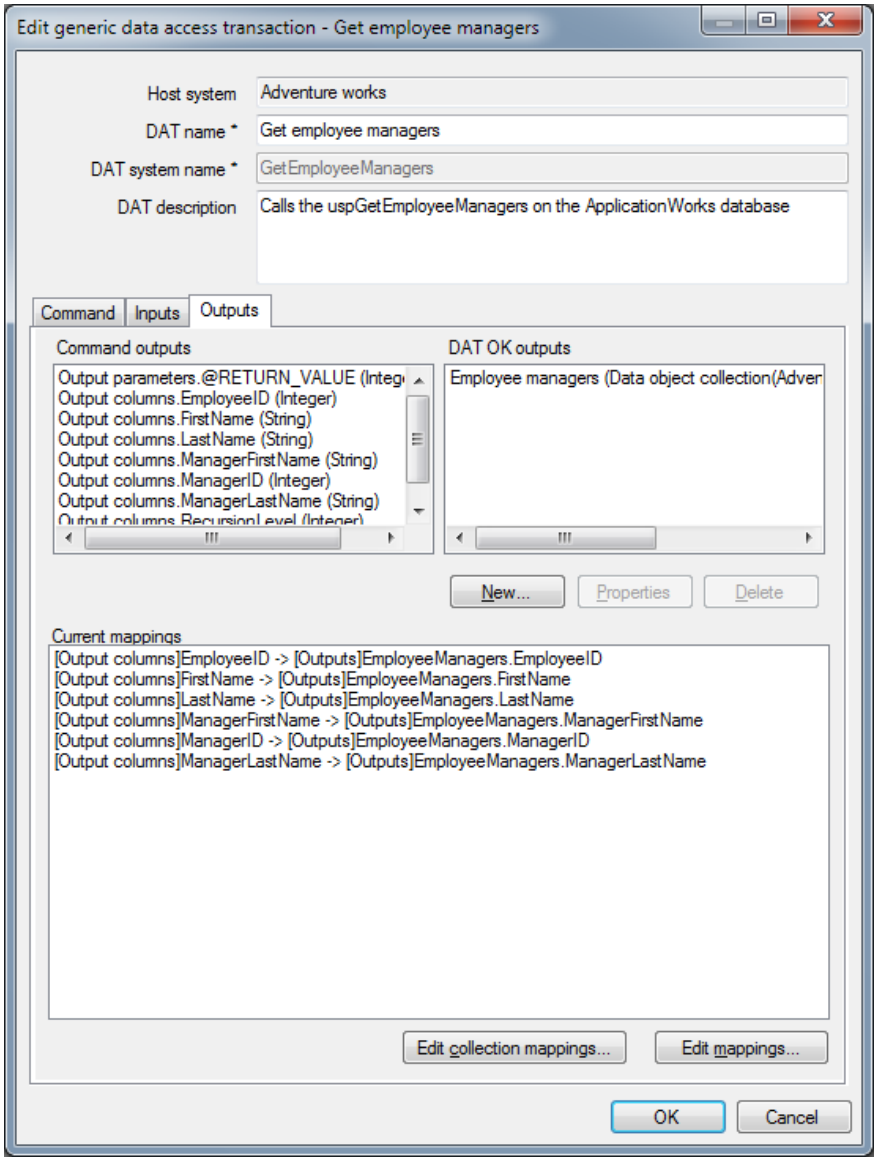
Simple mappings are created by clicking the **Edit mappings…** button. Mapping from a constant is possible when creating input mappings.

Collection mappings are created by clicking the **Edit collection mappings…** button. This button is only available if the command definition indicates the presence of an input collection **and** the Generic Access component supports this type of mapping.

2.4.1.2 Outputs tab

The Outputs tab displays the mappings from command outputs (shown with their corresponding Portrait Foundation data types) to DAT outputs.

Figure 3 – Example Outputs tab



DAT Outputs are managed using the **New…**, **Properties** and **Delete** buttons.

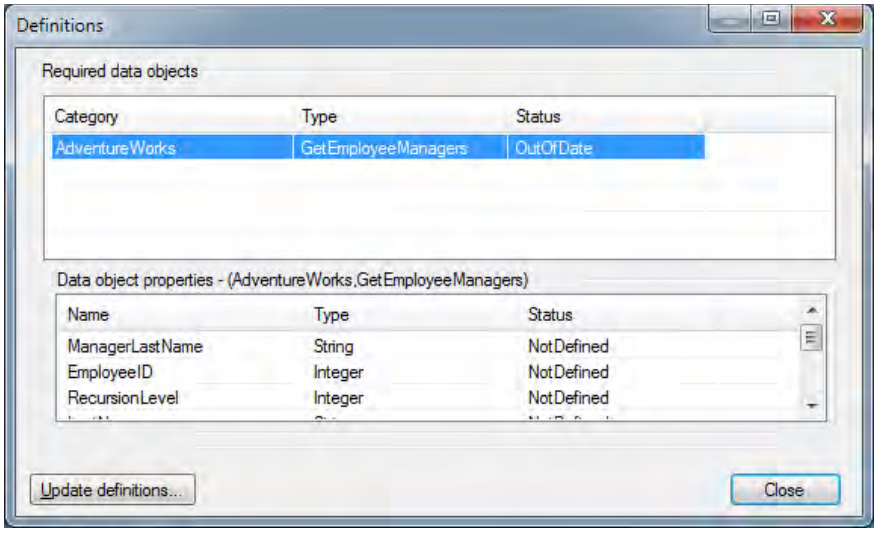
Simple mappings are created by clicking the **Edit mappings…** button.

Collection mappings are created by clicking the **Edit collection mappings…** button. This button is only available if the command definition indicates the presence of an output collection **and** the Generic Access component supports this type of mapping.

2.4.2 Data object definitions

Where a transaction editor is dealing in complex data types or collections, it may be useful to be able to define corresponding data objects. For example an output record set may need to be mapped to a collection of data objects whose properties correspond to the output columns. A standard dialog is provided for the purpose of creating or updating such data object definitions.

Figure 4 – Data object definitions dialog



This dialog provides the capability to create new data object definitions or update existing ones by selecting a data object and clicking the **Update definitions...** button. Note that the data object definitions that the dialog creates have the Data Object Category set to the External System's system name and the Data Object Type set to the Transaction's system name. The data object definitions generated by the data object definitions dialog are Portrait Foundation custom data objects and may be edited in the Portrait Foundation workspace in the normal manner.

You do not have to use these definitions in the mapping dialogs. For example, you may wish to map to/from an existing Portrait Data Object such as Party or Product. If you are planning to re-use a data object definition among several transactions (and therefore it may be inappropriate for the data object definition to be named after one of the transactions), you may prefer to create the data object definition manually.

2.5 Other editors

Standard Portrait Foundation editors, such as Generic Edit, are used wherever possible.

2.6 Deployment

Generic Access systems and transactions conform to the requirements of the Host Integration Framework and are deployed by the standard deployment mechanism.

3 Generic Database Access

Throughout this section, the term **database** is used to describe any source that can supply data in a tabular format and that can be accessed using standard SQL commands.

3.1 Capabilities

The GDA enables simple database operations to be defined within the Portrait Foundation Configuration Suite and to be executed on the Portrait Process Server, without the need for any code development. As an extension to Portrait's Host Integration Framework, it provides read and write access to any data source for which an OLE DB provider is available. In theory this means that virtually all data sources from a high-end DBMS (such as SQL Server, Oracle or DB2) to a text file can be supported (this also includes Excel spreadsheets, comma-separated files, Exchange servers and so on).

Through the GDA, the onfigure can specify SQL statements and stored procedures within the target database. Parameters to the SQL and stored procedures can be mapped from data within Portrait Foundation process models, and the outputs from the database can be mapped back into those models as scalar types, Portrait Data Objects or a combination of the two. If the output consists of multiple rows of data, the GDA constructs a collection of Portrait Data Objects, with each object representing one row.

Because of its generic approach, the GDA cannot offer full coverage of all functionality available in all data sources. Its capabilities are limited by the functionality supported by the underlying OLE DB provider. It is anticipated that most database access requirements, perhaps 80%, will be satisfied by the GDA. For the remainder that require more specialised database features it is necessary to develop individual Data Access Transactions using the Portrait SDK.

It should also be noted that the GDA is unlikely to provide optimal performance when accessing a data source. Whilst the performance should be satisfactory for most purposes, the use of a custom Data Access Transaction should be considered for performance-critical areas.

3.1.1 Providers

Note that there may be more than one OLE DB provider available for each system. For example, Microsoft, Oracle and third-party vendors have OLE DB providers for Oracle databases.

Each provider may exhibit different characteristics, such as database versions supported and functionality supported.

Because of the wide range of data sources and providers available, it is not possible to achieve complete coverage in testing of GDA components. Appendix A summarises the testing that has been performed.

3.1.2 Supported data types

Three different data type systems must be considered in the context of the GDA:

1. The native data types supported by the individual database system. For example, Oracle types include **VARCHAR2**, **NUMBER** and **DATE** whilst SQL Server types include **varchar**, **numeric** and **datetime**.
2. A standard set of types used within the OLE DB environment, such as DBTYPE_STR, DBTYPE_NUMERIC and DBTYPE_DBTIMESTAMP. These are described in more detail in Appendix C OLE DB data types on page 43. The purpose of these is to present a unified set of types to the OLE DB client

application, regardless of the underlying native database type system. Whilst all of the standard database types are also represented by OLE DB types, some databases may support types for which there is no OLE DB equivalent. It is not possible to read and write such data through OLE DB.

3 Portrait Foundation uses its own simplified set of scalar data types internally: Integer, String, Floating point number, Date/time, Boolean. These are described in more detail in Appendix B Portrait scalar data types on page 42.

As a data item passes through the GDA, it is converted between a Portrait Foundation internal type and an OLE DB type. The OLE DB provider converts between the OLE DB type and the native database type.

The GDA deals with converting between OLE DB types and Portrait Foundation internal types. For each OLE DB data type, where there is no directly corresponding Portrait Foundation data type that can contain all values of that type with no loss of data, it is necessary to choose a Portrait Foundation type to which to convert. There is a default conversion between these types (described in the table below) that is set up when a GDA System is created. You can then apply your knowledge of the underlying database to adjust the conversion for the System, for example, choosing to convert a database item of type DBTYPE_NUMERIC to a Portrait Foundation type of Integer because it is known that the database item always contains values that can be represented within a four-byte signed integer.

Table 1 – Data types supported by the Generic Database Adapter shows the set of OLE DB data types that are supported by the Portrait Foundation GDA and also shows the corresponding native database types for SQL Server and Oracle for reference, plus the Portrait Foundation types to/from which they are converted by default.

Table 1 – Data types supported by the Generic Database Adapter

OLE DB data type	Oracle native type	SQL Server native type	Portrait type
Numeric types			
DBTYPE_I1			**Integer**
DBTYPE_I2		**smallint**	**Integer**
DBTYPE_I4		**int**	**Integer**
DBTYPE_UI1		**tinyint**	**Integer**
DBTYPE_UI2			**Integer**
DBTYPE_UI4			**String**
DBTYPE_ERROR			**Integer**
DBTYPE_R4		**real**	**FloatingPointNumber**
DBTYPE_R8	**FLOAT**	**float**	**FloatingPointNumber**
DBTYPE_NUMERIC	**NUMBER**	**numeric** **decimal**	**String**
DBTYPE_VARNUMERIC	**NUMBER**		**String**
DBTYPE_I8		**bigint**	**String**
DBTYPE_UI8			**String**
DBTYPE_DECIMAL			**String**
DBTYPE_CY		**money** **smallmoney**	**String**
DBTYPE_BOOL		**bit**	**Boolean**
String types			
DBTYPE_STR	**CHAR** **VARCHAR2** **NCHAR** **NVARCHAR2** **ROWID**	**text** **char** **varchar**	**String**

OLE DB data type	Oracle native type	SQL Server native type	Portrait type
DBTYPE_WSTR		**nchar** **ntext** **nvarchar**	**String**
DBTYPE_GUID		**uniqueidentifier**	**String**
DBTYPE_BSTR			**String**
Date / time types			
DBTYPE_DBDATE			**DateTime**
DBTYPE_DBTIME			**DateTime**
DBTYPE_DBTIMESTAMP	**DATE**	**datetime** **smalldatetime**	**DateTime**
DBTYPE_DATE			**DateTime**
DBTYPE_FILETIME			**DateTime**

Some of the OLE DB types are *not* currently supported by the Portrait Foundation GDA because there is no meaningful conversion to an internal Portrait Foundation data type. These are shown in Table 2 – Data types not supported by the Generic Database Adapter, along with their Oracle and SQL Server equivalents.

Table 2 – Data types not supported by the Generic Database Adapter

OLE DB data type	Oracle native type	SQL Server native type
DBTYPE_BYTES	**LONG RAW** **RAW** **BLOB** **BFILE**	**varbinary** **timestamp** **binary** **image**
DBTYPE_EMPTY		
DBTYPE_NULL		
DBTYPE_IDISPATCH		
DBTYPE_VARIANT		**sql_variant**
DBTYPE_IUNKNOWN		
DBTYPE_ARRAY		
DBTYPE_BYREF		
DBTYPE_VECTOR		
DBTYPE_RESERVED		
DBTYPE_UDT		

3.1.3 Selectable type conversion

Since the input and output data for a GDA Command needs to be mapped in and out of a Portrait Foundation model using Portrait Foundation data types, it is necessary to convert the OLE DB data types of the GDA Command's parameters and outputs to Portrait Foundation data types.

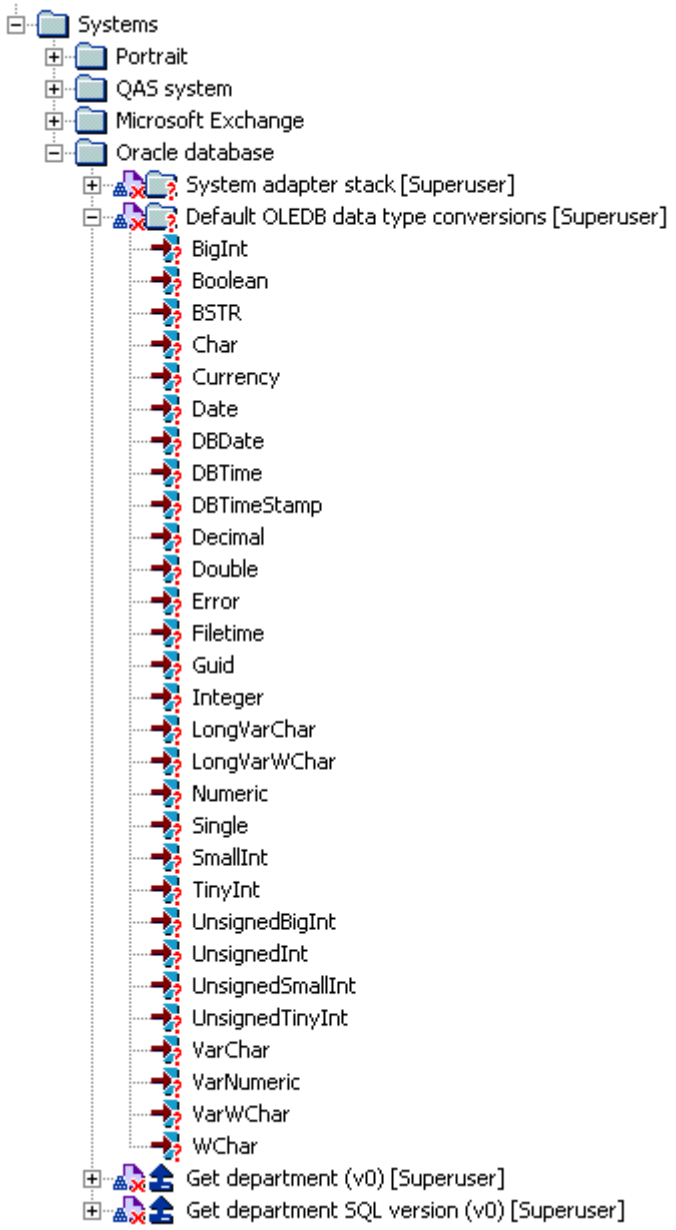
The default conversions between OLE DB data types and Portrait Foundation data types are shown in Table 1 – Data types supported by the Generic Database Adapter on page 13. Note that wherever data is converted between types, issues of scale, precision and format can arise. The default type conversions have been chosen to minimise the risk of data loss when converting from OLE DB data types to Portrait Foundation data types.

Whilst these default type conversions will be applicable for the majority of situations, you may find that there are situations where you wish to use different type conversions, and for this reason Portrait Foundation allows you to select the Portrait Foundation type to which you want an OLE DB data item to be converted. This selectable type conversion is provided in order to offer more flexibility in how database data items can be mapped to Portrait Foundation data items, however you are strongly advised to exercise caution when choosing alternative type conversions, especially when converting between types that have different underlying representations. (use the information in Appendix B Portrait scalar data types and Appendix C OLE DB data types as a guide). It is up to you to determine that the data being subjected to a conversion is compatible with the type conversions that are attempted. **In particular, you should note that conversion from fixed-precision numeric OLE DB types (such as Numeric, Decimal and Currency) to any Portrait Foundation type other than String is likely to lead to loss of precision.**

The actual type conversions are performed at runtime using standard .NET Framework type conversions.

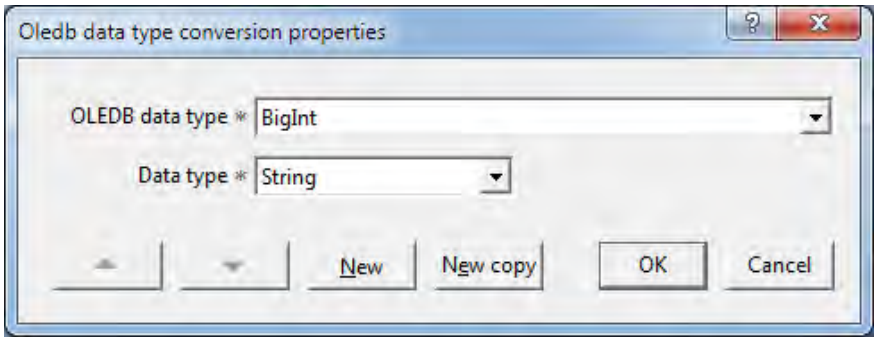
Default type conversions are held at the GDA system level in the Configuration Suite. When you create a GDA system, it is given the standard set of default type conversions. The figure below shows the location of this data in the configuration tree.

Figure 5 – Default OLE DB data type conversions



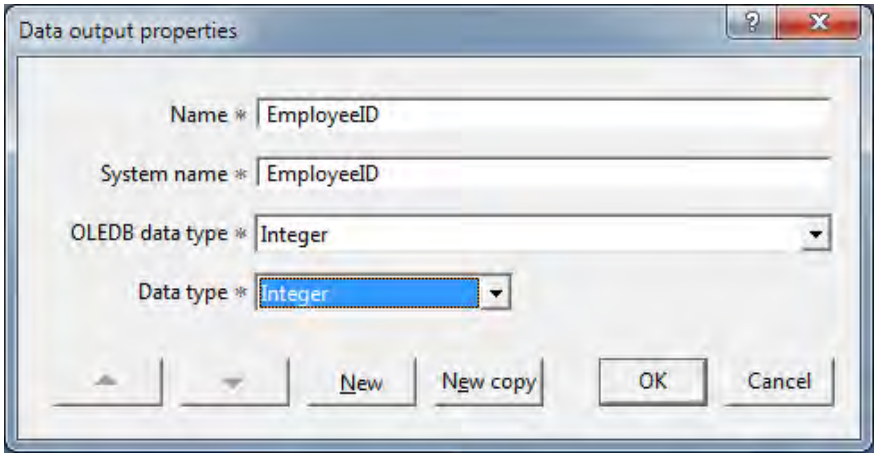
Each element in the set of default type conversions specifies the Portrait Foundation data type to and from which an OLE DB data type should be converted. This relationship can be modified by selecting an element in the set and using the generic properties dialog to edit it as shown in the figure below.

Figure 6 – Editing a default OLE DB type conversion



These default OLE DB data type conversions are used whenever the GDA transaction editor creates parameters or outputs automatically, for example when the user has selected a stored procedure by browsing from a database. The GDA transaction editor initialises the parameter or output with the Portrait Foundation data type specified for its OLE DB data type, but this can be overridden. If you wish to modify the type conversion for a particular output, for example, you can edit the properties of the output in the GDA transaction editor, by selecting it and using the generic properties dialog as shown in the figure below:

Figure 7 - Modifying the type conversion for an output

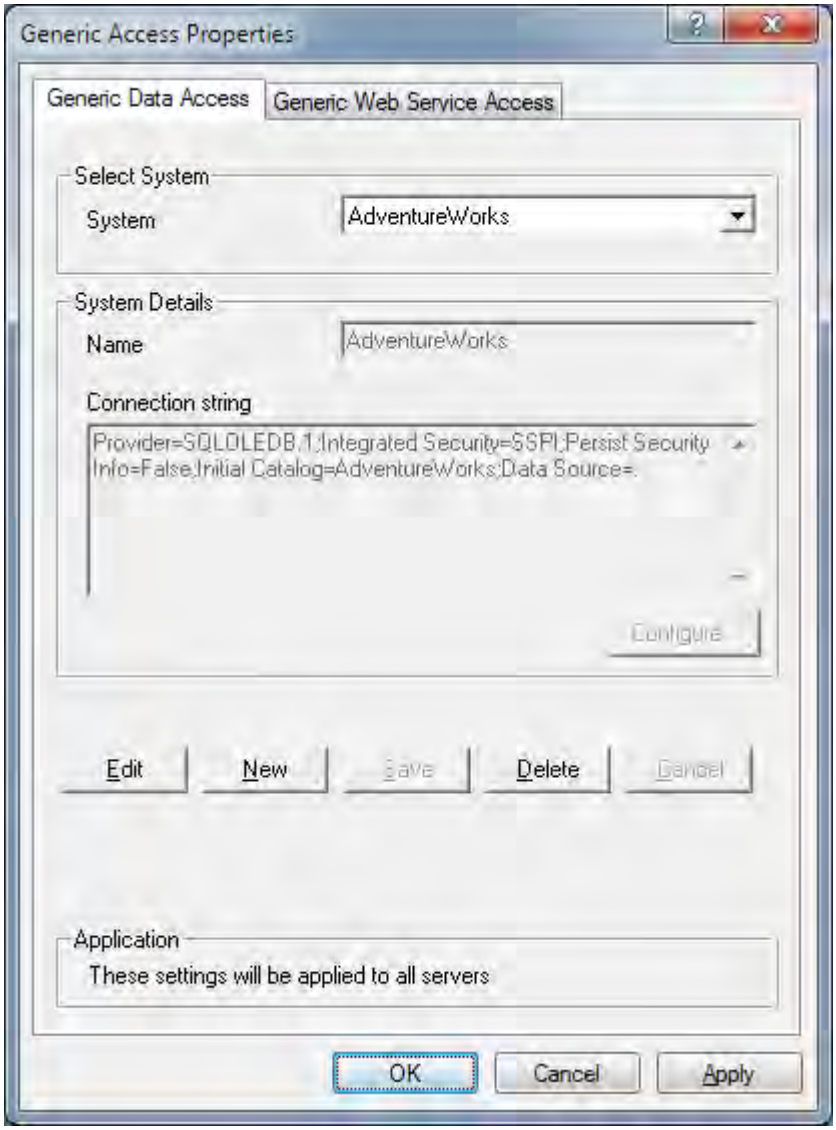


If you find that you need to modify the type conversion for a majority of instances of a particular OLE DB data type, you should consider modifying the default OLE DB type conversion as described before.

3.2 Management Console

Before any configuration may take place, each database to be targeted through the Generic Database Access components must first be defined through the Management Console.

Figure 8 – Management Console GDA System properties



3.2.1 Connection string

The **Configure...** button gives access to the standard Microsoft Data Link Properties dialog, which allows you to configure all aspects of the connection string.

Alternatively, the connection string may be edited directly in the **Connection string** textbox.

There may be occasions where it is necessary to setup several Generic Data Access systems in the Management Console that relate to the same physical database because the connection string contains both generic and provider-specific settings, whose values may need to differ for certain groups of transactions. For example, some transactions might need to run under one particular user account, others under another user account.

Users should consult their OLE DB provider documentation for advice on configuring the connection string. Some limited information for a variety of providers is available at: http://www.connectionstrings.com.

3.3 Configuration Suite

In the Host Integration Framework, the first step in configuring Data Access Transactions is to create an External System. In the Portrait Foundation workspace, all External Systems are created below the Systems node.

Once an External System node has been created, its child Data Access Transactions may be added.

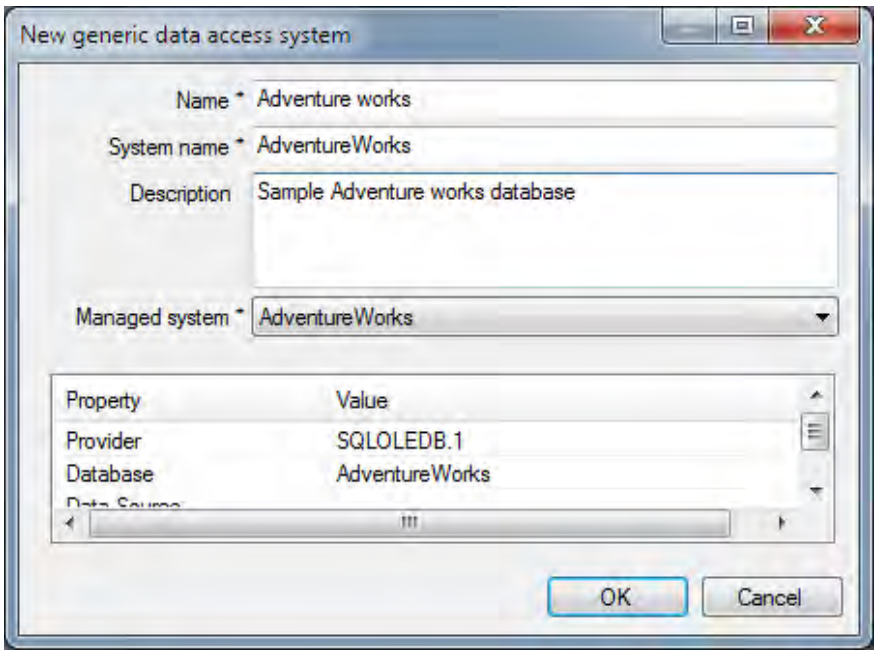
Creating GDA transactions follows the same pattern.

3.3.1 GDA System Editor

In order to create an External System that uses GDA components, the **New generic data access system...** context menu item must be selected (as opposed to **New system...** for a standard HIF External System). This launches the GDA System Editor.

The GDA System Editor is responsible for adding/editing a GDA system in the workspace. The primary function of the editor is selecting one of the managed systems defined in the Management Console.

Figure 9 – GDA System Editor



It is possible to configure a number of logical GDA systems that use the same physical managed system defined in the Management Console.

3.3.2 GDA Transaction Editor

The GDA Transaction Editor is responsible for adding/editing a transaction in the workspace. The editor consists of one area containing common transaction information irrespective of the technology, and a tabbed area containing the following tabs:

- **Command** – This is where the database command configuration is defined.
- **Inputs** – This tab is used to define the DAT inputs and to map those inputs onto database command input parameters.
- **Outputs** – This tab is used to define the DAT outputs and to map database command output parameters and record sets onto those DAT outputs.

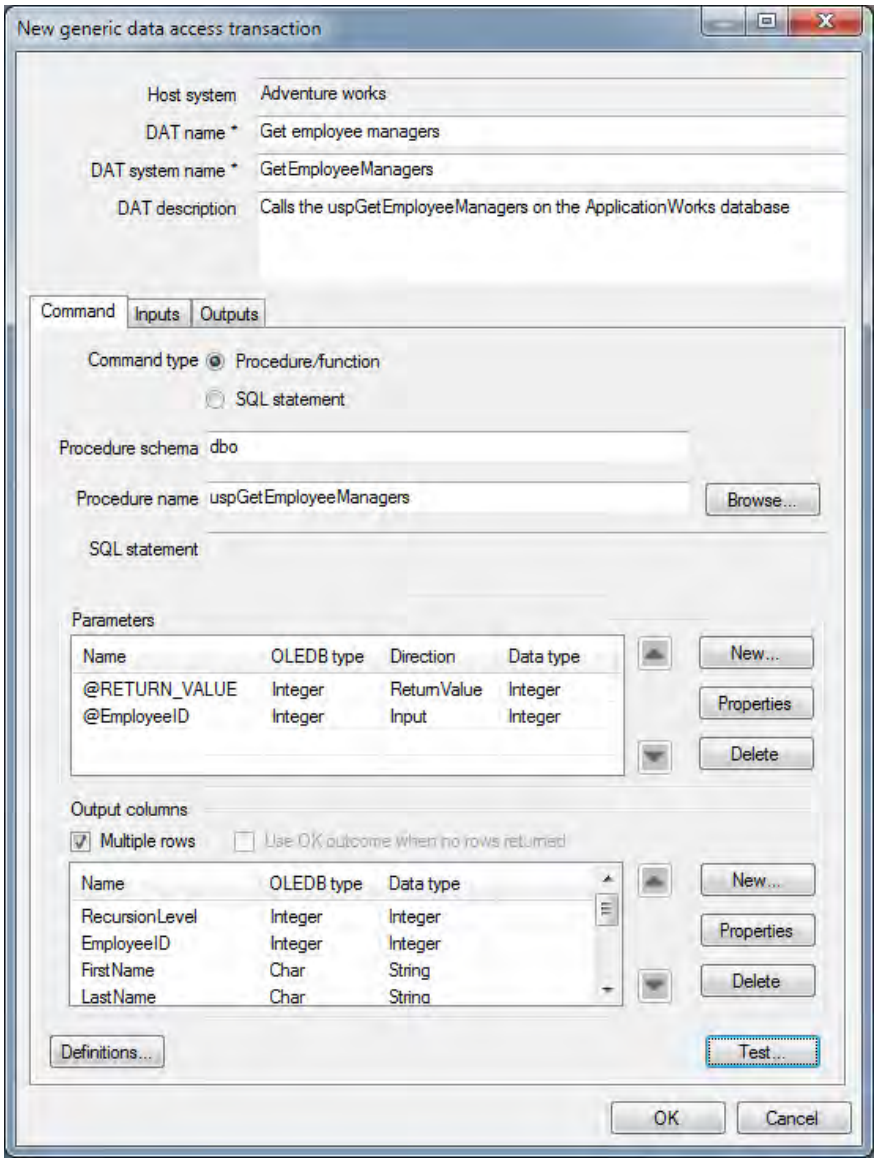
3.3.3 GDA Transaction Editor – Command tab

This is where the database command is defined. The command comprises:

- **Command type** – A command may be either a SQL statement or a stored procedure.
  - **SQL statement** – SQL statements must be entered manually. The SQL statement may contain provider-specific syntax. Where a run-time parameter is required, the parameter delimiter is the standard ODBC '?' character.
  - **Procedure/function** – Stored procedure/function names may be entered manually but, if there is a connection to the database available and the provider supports catalogue browsing, then the database catalogue may be browsed and a procedure/function selected.
- **Parameters** – Details of the parameters required to execute the SQL statement or stored procedure must be captured. In the case of a stored procedure, it may be possible to use the catalogue browse function to retrieve details of the parameters.
- **Output columns** – Details of any returned record set must be captured. Again this may be entered manually but, if there is a connection to the database available, the SQL statement or stored procedure/function may be executed with sample values. Any returned record set will then be examined and the output columns discovered.

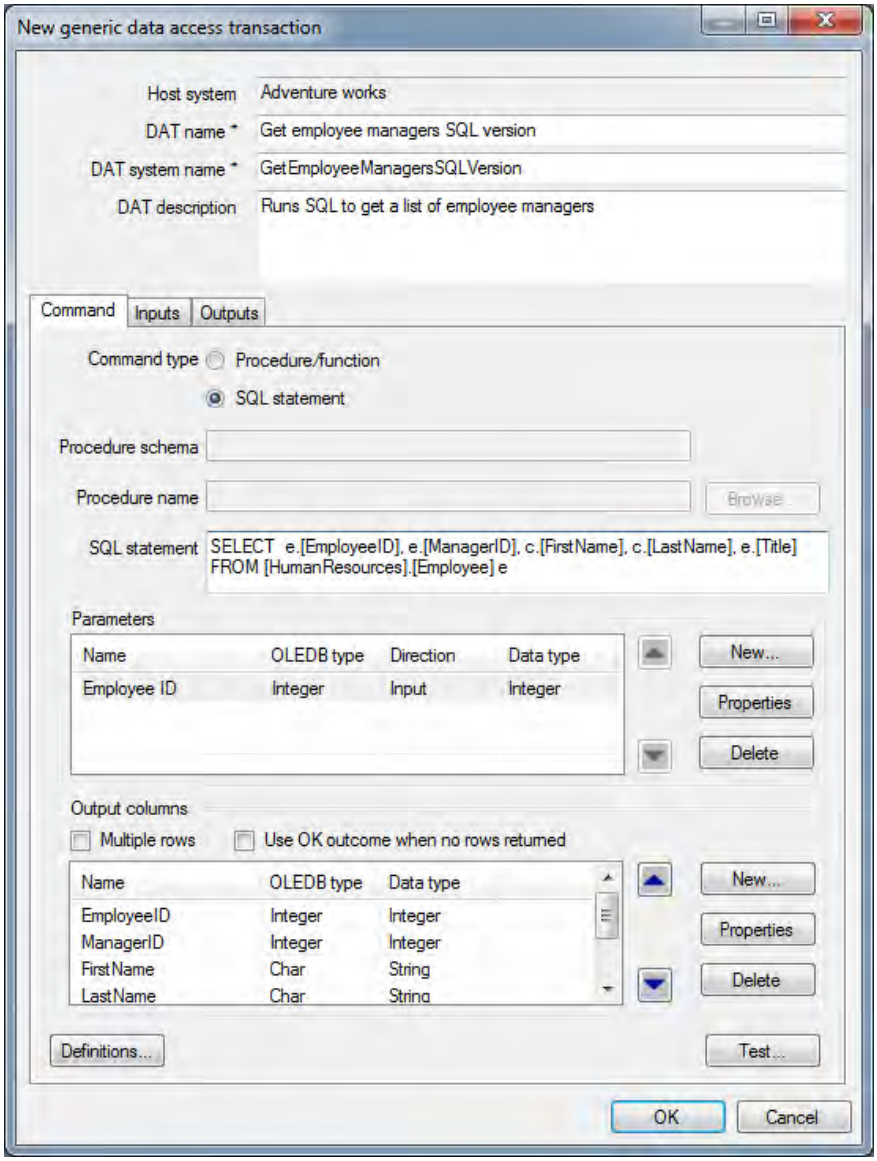
Consider the example Oracle stored procedure in Appendix B. Figure 10 shows the GDA Transaction Editor, containing the appropriate stored procedure configuration.

Figure 10 – Example transaction editor for an ORACLE stored procedure.



The configuration for a similar SQL statement would be:

Figure 11 – Example GDA Transaction Editor SQL statement



As described above, it is possible to parameterise the SQL statement through the use of bind variables in the WHERE clause. However, all other aspects of the SQL statement must be fixed at configuration time. For example, the column names and ORDER BY clause, if required, must be fixed. Whilst this is a minor limitation, it should still be possible to handle the vast majority of requirements for database access.

3.3.3.1 Parameters and output columns

The SQL statement or stored procedure/function's parameters and output columns are managed via their respective **New...**, **Properties** and **Delete** buttons. The **New...** and **Properties** buttons launch the standard Portrait Generic Edit dialog.

Note that when configuring stored functions, the return value parameter is the first parameter. For Oracle the return value parameter name is not significant.

3.3.3.2 Multiple rows

This checkbox should be selected if you wish to return each row of the resulting record set as a data object within a data object collection.

If the command is expected to return a single row of data (or you know that you are only interested in the first row returned) **and** you do **not** wish to return the row as a data object within a collection then you should clear this checkbox.

The state of this checkbox affects whether the **Use OK outcome when no rows returned** checkbox is enabled and whether the **Edit collection mappings…** button is available on the Outputs tab (see sections 2.4.1.2, 3.3.3.3 and 3.3.5)

3.3.3.3 Use OK outcome when no rows returned

This checkbox will only be enabled if the Multiple rows checkbox is cleared.

When checked, the DAT will complete on the OK outcome regardless of whether rows are returned. When no rows are returned, the outputs of the DAT will be empty.

When the checkbox is cleared, the DAT will complete on the FAIL outcome when no rows are returned.

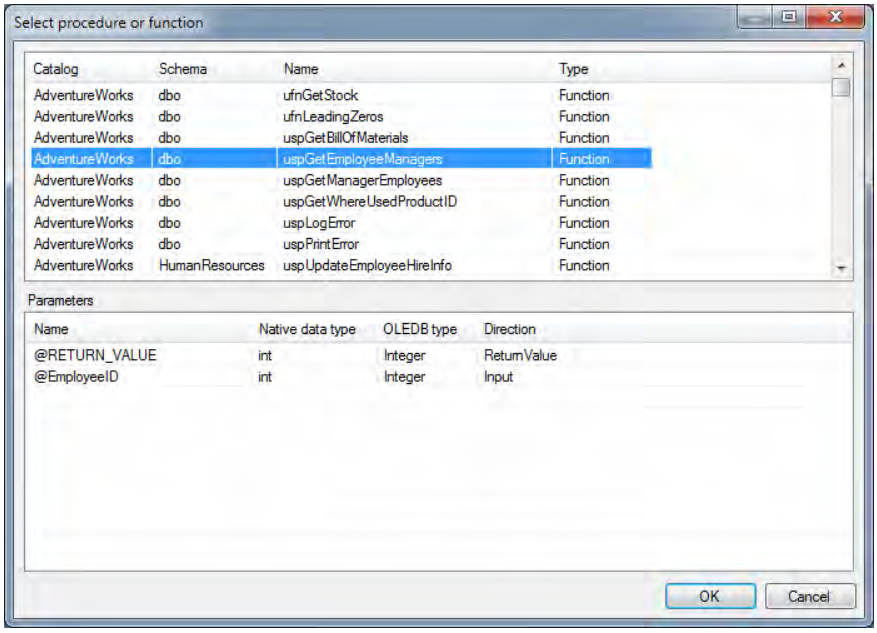
3.3.3.4 Definitions

This button launches the Data Object Definitions dialog, see section 2.4.2.

3.3.3.5 Browse

It is possible to browse a data source catalogue for stored procedures (if the selected OLE DB provider supports this feature). Clicking the **Browse…** button launches the following dialog (there may be a delay while the catalogue is queried):

Figure 12 – Browsing for stored procedures

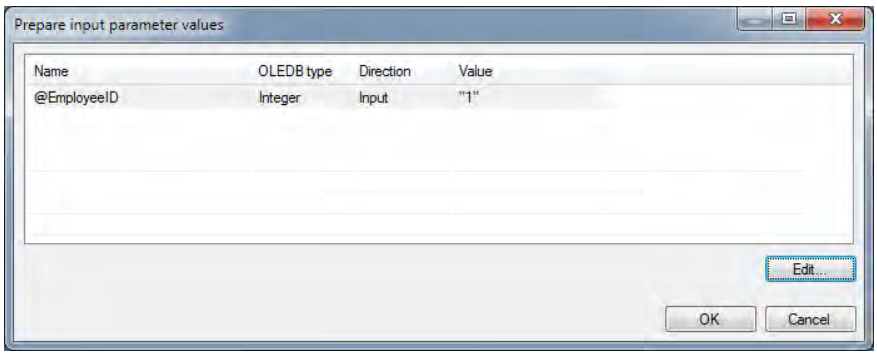


Clicking **OK** replaces the configuration in the main form with the selected procedure and its parameters. You will be asked to confirm the change.

3.3.3.6 Test

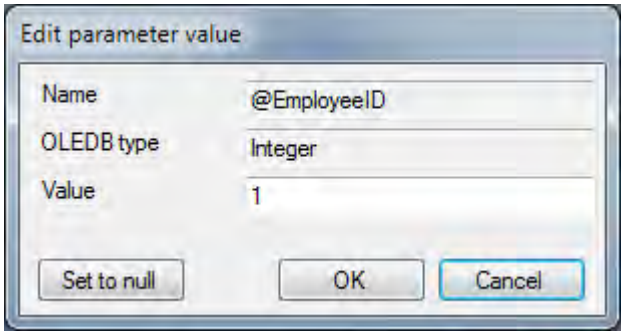
It is possible to execute a stored procedure or SQL statement in order to discover details of its output record set. Clicking the **Test…** button launches the following dialog:

Figure 13 – Supplying input parameter values



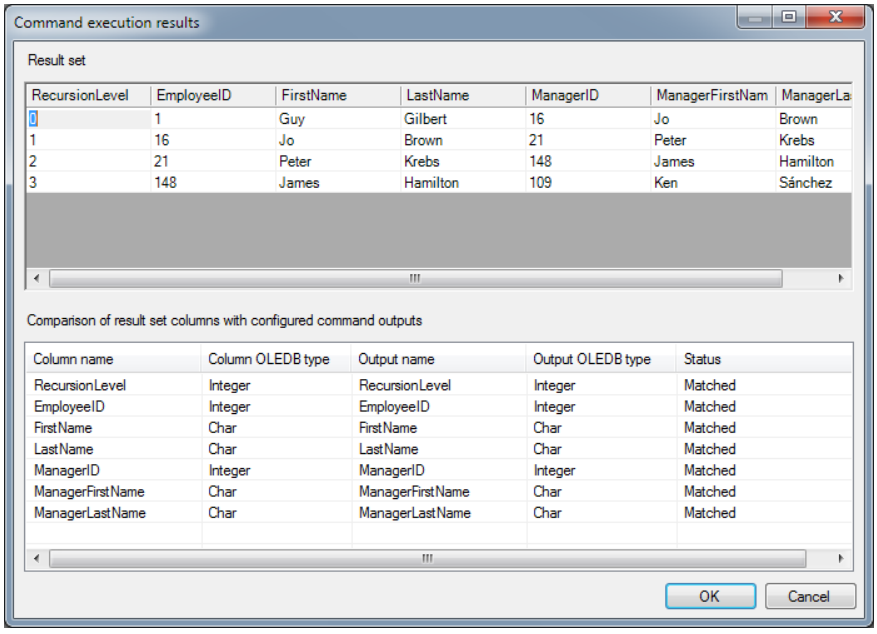
A list of input parameters is displayed. Selecting a parameter and clicking the **Edit...** button allows the value of the input parameter to be specified:

Figure 14 – Specifying an individual value



Once all required values have been supplied then the SQL statement or procedure/function may be executed by clicking **OK** on the Parameter values dialog. When execution completes successfully, the returned record set and details of the returned columns are displayed:

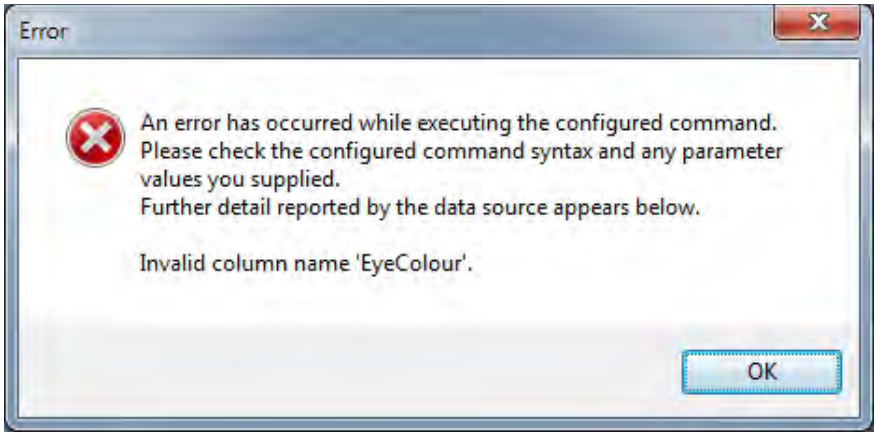
Figure 15 – Returned record set details



Click **OK** to accept the discovered column information into the main transaction configuration dialog.

If, for any reason, command execution fails, the editor displays a message like this:

Figure 16 – Command execution error



The text at the bottom of the message comes from the data source and is the same message you would see in an interactive database query tool such as Microsoft Query Analyzer or Oracle SQL*Plus. The example shown results from executing a SQL statement with an incorrect column name.

3.3.4 GDA Transaction Editor – Inputs tab

The main features of this tab are described in section 2.4.1.1.

In the example above, there is only a single input EmployeeID. This must be mapped from either a DAT input (of type Integer, assuming the default type conversions are used) or a constant. This is achieved by clicking the **Edit mappings...** button.

3.3.5 GDA Transaction Editor – Outputs tab

The main features of this tab are described in section 2.4.1.2.

In the example above, there are no **Output** (or **InputOutput**) parameters and the SQL record set is assumed to consist of multiple rows of data. There is therefore only a need to create mappings for the output columns. This is achieved by clicking the **Edit collection mappings...** button.

If the multiple rows checkbox was cleared on the Command tab (that is, only a single or first row is of interest) then each output column can be mapped to an individual output item or data object property using the **Edit mappings...** button.

When only a single row is expected, the DAT will fail at runtime if no records are returned. This default behaviour can be overridden by checking the 'Use OK outcome when no rows returned' check box on the Command tab. In this situation, the DAT will complete on the OK outcome and all outputs will be empty.

3.4 Usage guidelines

- Do:
  - Use stored procedures and functions
    - This reduces network bandwidth requirements
    - These are compiled
    - These execute on the most appropriate platform (the database server, where the data resides)
  - Validate parameter values
    - Since Portrait Foundation stores data in a reduced number of data types, such as four-byte signed integer for all integral values, there is the possibility of overflow when converting to OLE DB types such as DBTYPE_I2.
- Don't:

- Retrieve large record sets, this causes
  - High network bandwidth requirements
  - Large Portrait Data Object Collections to be built and transferred to the model context.
  - Complex post processing and model iteration
- Retrieve unused columns, this compounds the first two sub-items above.

Despite extensive facilities for testing at configuration time, it is still possible to deploy GDA configurations that can experience problems at runtime, due to features of the data being processed or changes to the database objects being accessed. When Portrait Foundation detects an error, the detail of the error is recorded in the Portrait Foundation error log. The table below describes some runtime conditions that may arise and describes some possible causes. (This table is intended to be indicative of potential error conditions rather than exhaustive.)

Table 3 – Troubleshooting

Condition	Possible cause
DAT fails with an OleDb Exception error	The SQL statement or stored procedure contains an error or one of the parameter values passed to it is invalid in some way (such as empty or out of range)
DAT fails with an Invalid Cast Exception error	Conversion from one numeric type to another numeric type where the value is too large to be represented as the destination type. For example: conversion from Portrait FloatingPointNumber to OLE DB Decimal where the value being converted is greater than 10^{38}-1
DAT fails with an Overflow Exception error	Conversion from a string to a numeric type where the value is too large to be represented as the destination type. For example: conversion from Portrait String to OLE DB UnsignedTinyInt where the value is “257”
DAT fails with a Format Exception error	Conversion from a string to a numeric type where the value is not in a format that can be converted to the destination type. For example: conversion from Portrait String to OLE DB DBTimeStamp where the value is “NotAValidDate”
DAT succeeds, but precision is lost in numeric data	Conversion from a fixed-precision type (such as Numeric, Decimal and Currency OLE DB types) to a floating point type such as Portrait FloatingPointNumber
DAT succeeds, but string data is truncated	Calling a stored procedure with a parameter defined as char(*N*) and passing a string with more than *N* characters. Some OLE DB providers (such as SQLServer) silently truncate the data to *N* characters.

3.5 Known limitations and issues

The GDA is a re-usable component that provides a high degree of configurability and relative ease of use. This generic functionality must be traded-off against the more complex data processing that is possible when writing custom code for each database interaction.

A number of limitations are imposed by the underlying data access mechanism (OLE DB) and Portrait Foundation itself imposes some additional limitations. Some limitations affect specific database management systems only, whilst others are more general.

A number of known limitations are described below, along with some workarounds. It is possible that further limitations will be discovered as different data sources are tested.

3.5.1 Stored procedure names must be unique

This limitation may affect Oracle data sources and any others that allow multiple stored procedures to share a common name.

In some database management systems it is possible to **overload** stored procedures such that several procedures share a common name. For example, Oracle supports overloading of procedure names by variation in the procedure signature, that is, multiple procedures with the same name that vary only by their signature (list of parameters).

Overloaded names are not suitable for use with GDA transactions because the GDA configuration editor and runtime need to be able to query the database catalogue for the parameters of a stored procedure, identifying the stored procedure by name. (There is no unique identifier for an instance of a procedure available through OLE DB.)

A workaround is to re-implement the stored procedure with different names for the different variations.

3.5.2 Only one result set can be processed

This limitation may affect any data source that can return multiple result sets.

If a stored procedure returns more than one result set, the GDA can only process the first one. The GDA transaction configuration editor and GDA runtime currently support only one result set due to the additional complexity that would be required to configure multiple result sets.

Possible workarounds are to re-implement a procedure that returns N record sets as N procedures that each return one record set or, alternatively, create N procedures, each of which calls the original procedure, but returns only one of the result sets.

3.5.3 TABLE return type not supported

This limitation may affect Oracle data sources.

When returning a record set from a stored procedure, Oracle permits the record set to be declared as a TABLE. However, TABLE is not supported by OLE DB and consequently it is not supported by the GDA. Instead, OUT parameters that are record sets must be declared as REF CURSOR. (See Oracle OLE DB documentation for further information on the use of REF CURSOR).

A possible workaround is to re-implement stored procedures that return TABLEs to return REF CURSORs instead. Alternatively, existing procedures could be wrapped by procedures having parameters compatible with OLE DB. If database changes are not permitted, it may be possible to write custom Data Access Transactions for each database interaction using a technology (other than OLE DB) that supports TABLEs as OUT parameters.

3.5.4 Public synonyms are not supported

This limitation may affect Oracle data sources and any others that support synonyms.

In some database management systems it is possible to define **public synonyms** as alternative names for database objects such as stored procedures. Public synonyms are typically used to provide a simple name to represent an object within a user schema that would otherwise require the use of a fully qualified name, which might be complex.

The GDA transaction configuration editor is unable to access synonym information within the database because this information is not exposed through OLE DB providers.

The workaround is to define the GDA transaction in terms of the actual (qualified) object name rather than in terms of a synonym. Where previously a database user may have had access to an object owned by a different user through rights granted on a synonym, it may now be necessary to explicitly grant access rights on the object itself.

3.5.5 Large character object types not supported in command parameters

This limitation may affect Oracle data sources and any others that support large character object data types. In the case of Oracle, these data types are LONG, CLOB and NCLOB.

Some database management systems allow the storage of large character data objects, with sizes in gigabytes. Using these data types in parameters to stored procedures or SQL statements requires techniques that are specific to the various database management systems, so cannot be implemented in a generic component.

It may be possible to write custom Data Access Transactions using non-generic techniques to handle such data types, for example by streaming data to and from files.

4 Generic Web Service Access

4.1 Capabilities

The GWSA enables Web service definitions (in WSDL documents) to be interpreted by the Portrait Configuration Suite and the selected SOAP operations to be executed on the Portrait Process Server, without the need for any code development. As an extension to Portrait's Host Integration Framework, it provides interoperability with most SOAP-based XML Web services.

Through the GWSA, the onfigure can select SOAP operations published in a WSDL document. The GWSA uses .Net Framework classes to generate a client proxy which exposes each SOAP operation as a method on a .Net class. Input parameters for the methods can be mapped from data within Portrait Foundation process models, and the output parameters and return values from the method can be mapped back into those models as appropriate scalar types, Portrait Data Objects or Portrait Data Object Collections.

Because of fixed conversions between Portrait Foundation data types and .Net method parameters and return types, SOAP operations which involve variable message content (such as **xsd:any**) cannot be invoked. It is anticipated that many Web services will be supported, perhaps 80%, will be satisfied by the GWSA. For the remainder that require more specialised message processing it is still necessary to develop individual Data Access Transactions using the Portrait SDK. (A future version of the GWSA may support the configuration of plug-in type converters to improve compatibility.)

Since the .Net client proxy is compiled and cached by the GWSA, it should provide reasonable performance when accessing a Web service. However where there is significant pre- or post-processing of the data exchanged in the SOAP operation, this must be performed outside the GWSA. In this situation, the use of a custom Data Access Transaction should be considered for performance-critical areas.

4.1.1 Supported data types

Two different type systems must be considered in the context of the GWSA:

1. The .Net types created when the WSDL document is interpreted and a client proxy generated.
2. Portrait Foundation uses its own simplified set of scalar data types internally: Integer, String, Floating point number, Date/time, Boolean. These are described in more detail in 'Appendix B Portrait scalar data types' on page 42.

As a data item passes through the GWSA, it is converted between a Portrait Foundation internal type and a .Net type.

Table 4 shows the set of .Net data types that are supported by the Portrait Foundation GWSA and also shows the corresponding Portrait Foundation data type.

Table 4 – Data types supported by the Generic Web Service Adapter

.Net data type	Portrait data type
Numeric types	
System.Single	**FloatingPointNumber**
System.Double	**FloatingPointNumber**
System.Decimal	**FloatingPointNumber**
System.Int8	**Integer**

.Net data type	Portrait data type
System.Int16	**Integer**
System.Int32	**Integer**
System.Int64	**Integer**
System.Uint8	**Integer**
System.Uint16	**Integer**
System.Uint32	**Integer**
System.Uint64	**Integer**
String types	
System.Char	**String**
System.String	**String**
System.Guid	**String**
Date / time types	
System.DateTime	**DateTime**
System.TimeSpan	**DateTime**
Boolean types	
System.Boolean	**Boolean**
Other types	
Non-System Class	**DataObject**
Array	**DataObjectCollection1**

Note that wherever data is converted between types, issues of scale, precision and format can arise. It is up to the onfigure to determine that the data being subjected to a conversion is compatible with the type conversions that are attempted. Failure to do so will be manifested as a failure in the DAT at runtime.

No other types are *currently* supported by the Portrait GWSA because there is no obvious conversion to an internal Portrait Foundation data type. A future version of the GWSA may support the configuration of plug-in type converters to improve data type conversion capabilities. For example, it may become possible to convert from **System.Xml.XmlNode** to **DataObject("*", "*")**.

4.1.2 Request Tokens and SOAP headers

Every request that arrives at the web tier of Portrait Foundation is allocated a Request Token that is passed through the system and may be logged along with performance data at various levels within the system. This Request Token takes the form of a GUID.

The current Request Token is made available to called web services in a Soap header called *StateHeader*. Processing of this Soap Header by the called web

[1] Note that a collection of a primitive type is converted to a **DataObjectCollection** with Category 'General' For example, **System.String[]** is converted to **DataObjectCollection**("General", "String")

service is optional, but if they wish to receive the Request Token in order to log it or pass it on further they can do so as follows:

```
using System.Web.Services.Protocols;

// declare a class to represent the Soap header
public class StateHeader : SoapHeader
{
    public string requestToken;
}

// add a public member variable
public StateHeader StateHeaderValue;
```

Then in your constructor, assign the member variable:

```
StateHeaderValue = new StateHeader();
```

And finally, modify the declaration of your web service method(s) by adding the following attribute:

```
[SoapHeader("StateHeaderValue", Direction=SoapHeaderDirection.In)]
```

Within the body of the method you can then access the following to get the current Request Token:

```
string requestToken = StateHeaderValue.requestToken;
```

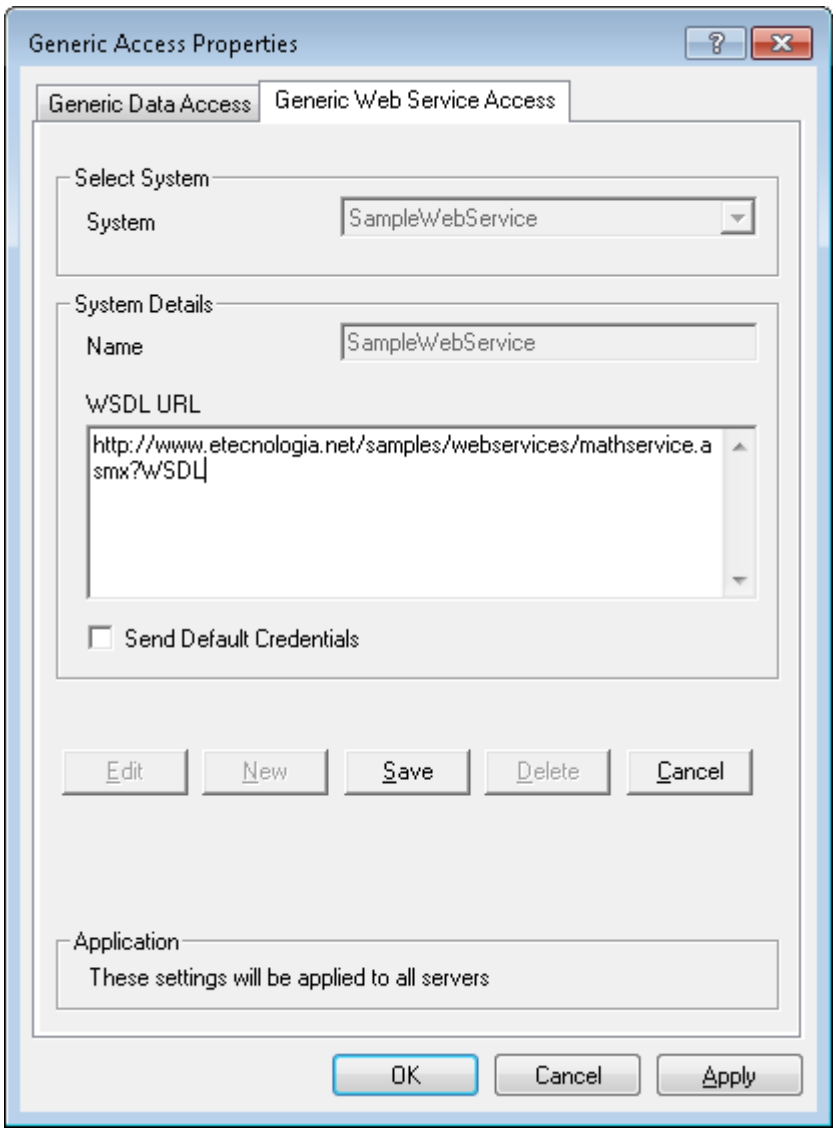
GWSA and SOAP headers

Note: SOAP headers are handled differently in the WCF-compatible GWSA and in the backwards-compatible (ASP.NET) version. See section 4.5.2 'SOAP headers'.

4.2 Management Console

Before any configuration may take place, each Web service to be targeted through the Generic Web Service Access components must first be defined through the Management Console.

Figure 17 – Management Console GWSA System properties



4.2.1 WSDL URL

Examples of URLs include:

- C:\temp\x.wsdl – a local file
- http://tempuri.org/webservice.wsdl – a file downloaded from a web server
- http://tempuri.org/webservice.asmx?WSDL – a WSDL document dynamically generated by a .Net Web service

Note that the **file:** and **res:** protocols are not supported.

4.2.2 Send default credentials

By checking the **Send default credentials** checkbox, you are selecting that calls to the web service will use the credentials of a windows user for authentication. This means that the service call will be authenticated against either the logged on user or the account running the Foundation ServiceHost service, depending on whether the web service is called from the Configuration Suite or via a process model. Please note that Basic Authentication (using a Base64-encrypted Username and Password) is not supported in GWSA. The above form of Windows Authentication is the only authentication version currently available.

4.3 Configuration Suite

In the Host Integration Framework, the first step in configuring Data Access Transactions is to create an External System. In the Portrait Foundation workspace, all External Systems are created below the Systems node.

Once an External System node has been created, its child Data Access Transactions may be added.

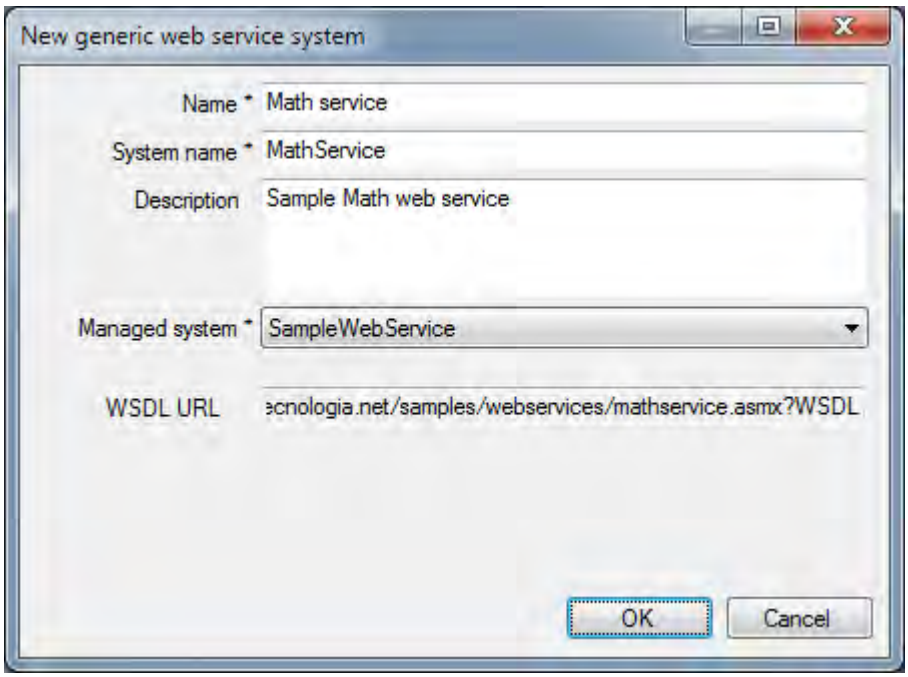
Creating GWSA transactions follows the same pattern.

4.3.1 GWSA System Editor

In order to create an External System that uses GWSA components, the **New generic web service system...** context menu item must be selected (as opposed to **New system...** for a standard HIF External System). This launches the GWSA System Editor.

The GWSA System Editor is responsible for adding/editing a GWSA system in the workspace. The primary function of the editor is selecting one of the systems defined in the Management Console.

Figure 18 – GWSA System Editor



It is possible to configure a number of logical GWSA systems that use the same physical system defined in the Management Console.

4.3.2 GWSA Transaction Editor

The GWSA Transaction Editor is responsible for adding/editing a transaction in the workspace. The editor consists of one area containing common transaction information irrespective of the technology, and a tabbed area containing the following tabs:

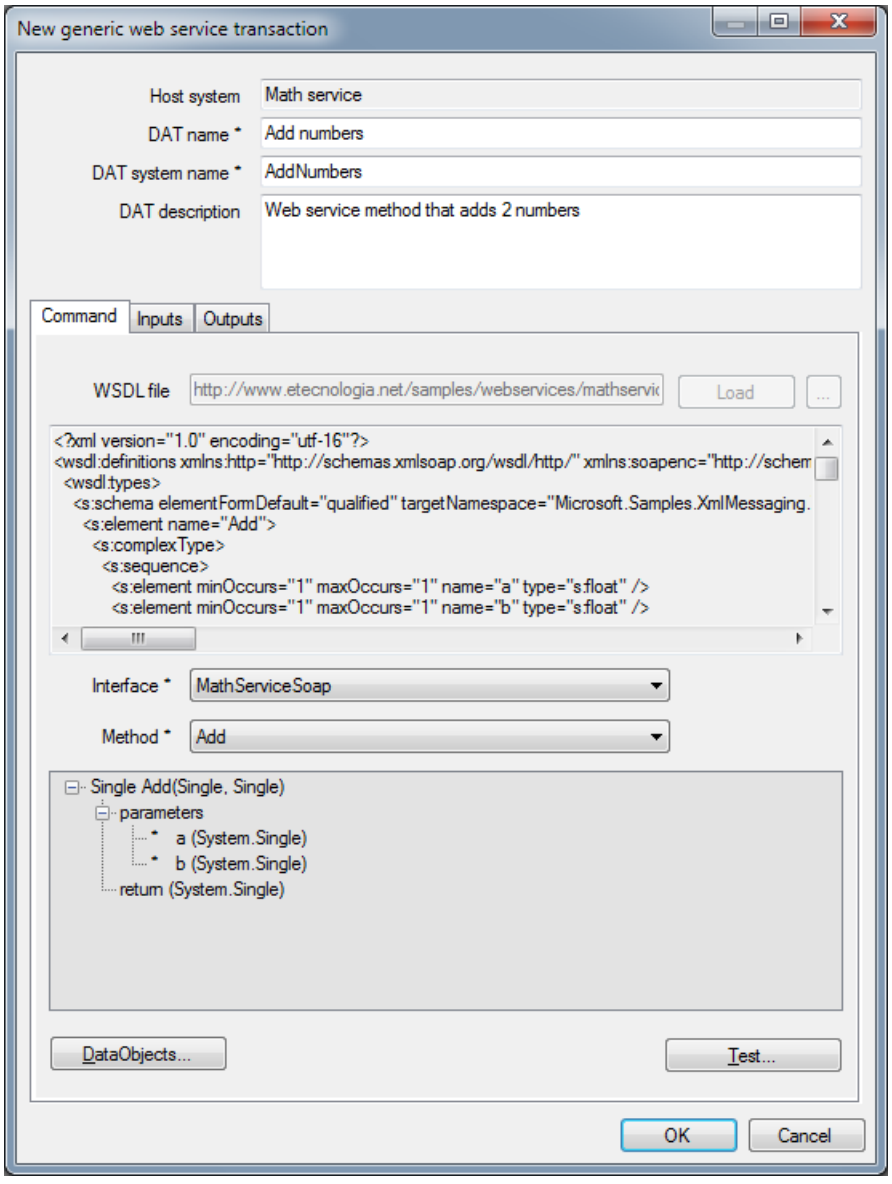
- **Command** – This is where the Web service-specific command configuration is defined.
- **Inputs** – This tab is used to define the DAT inputs and to map those inputs onto Web service client proxy input parameters.
- **Outputs** – This tab is used to define the DAT outputs and to map Web service client proxy output parameters and result value onto those DAT outputs.

4.3.3 GWSA Transaction Editor – Command tab

This is where the Web service-specific command is defined. The command comprises:

- **Interface** – The service interface selected from those listed in the WSDL document
- **Method** – The method (operation) selected from those available within the above service

Figure 19 – Example transaction editor for a Web service operation.



4.3.3.1 Data Objects

This button launches the Data Object Definitions dialog, see section 2.4.2.

Note it is most efficient to map complete complex types exposed as .Net classes to equivalent Portrait Data Objects (or collections) generated via the Data Object Definitions dialog. Constructing or decomposing such complex types using multiple mappings of individual properties imposes a significant performance penalty. This should be avoided unless only a small portion of the complex type is to be populated or extracted.

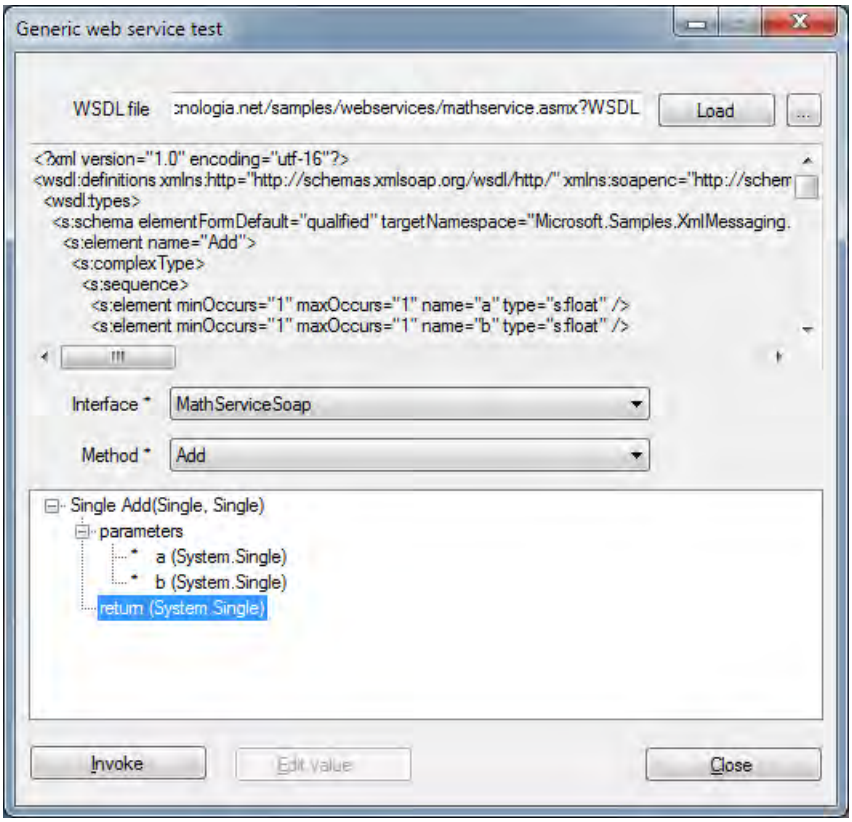
Nested Data Objects

Data Object Definitions dialog may fail to create nested Data Objects. If this happens, create as much as possible in Data Object Definitions dialog and then go to the Data Objects section in the Configuration Suite and find the incomplete Data Objects (the Category is the web service interface name, e.g. 'GWSA IsampleService'). Open an incomplete Data Object and manually set any Data Object nested within it using 'Edit properties', selecting the interface name as Category e.g. 'IsampleService'.

4.3.3.2 Test

It is possible to execute a SOAP operation in order to test its behaviour. Clicking the **Test...** button launches the following dialog:

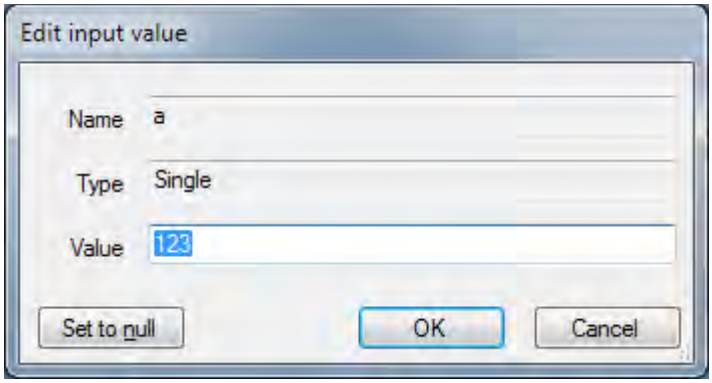
Figure 20 – Testing a SOAP operation



The test dialog allows a different WSDL URL to that defined in the Management Console to be specified and loaded (you may wish to configure against local WSDL files and test against a live service). The WSDL URL may be entered manually or click the **...** button to browse for a local file. Click the **Load** button to load the WSDL and generate a client proxy.

All input parameters are marked with an asterisk, '*'. Values for these inputs are supplied by double-clicking the parameter. To launch the **Edit input value** dialog:

Figure 21 – Specifying an individual value

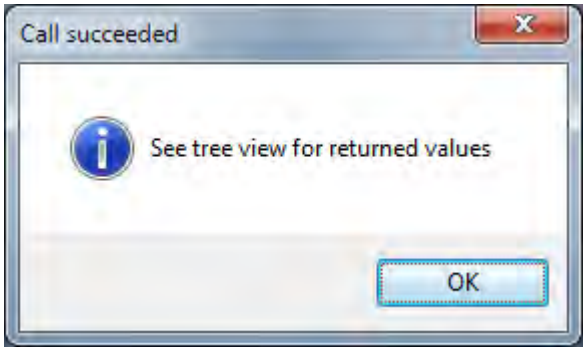


Note that where an input parameter contains an array (potentially nested within a complex type), the number of array elements must be specified first. Double-clicking the array item launches the **Edit input value** dialog, as above, and the number of elements must be entered. When the value is confirmed by clicking **OK**, the appropriate number of child elements will be generated in the signature and values for the element properties may then be supplied.

Also note that **String** values appear to have child elements (a numeric length and char array) however their value is still supplied as a single string entry.

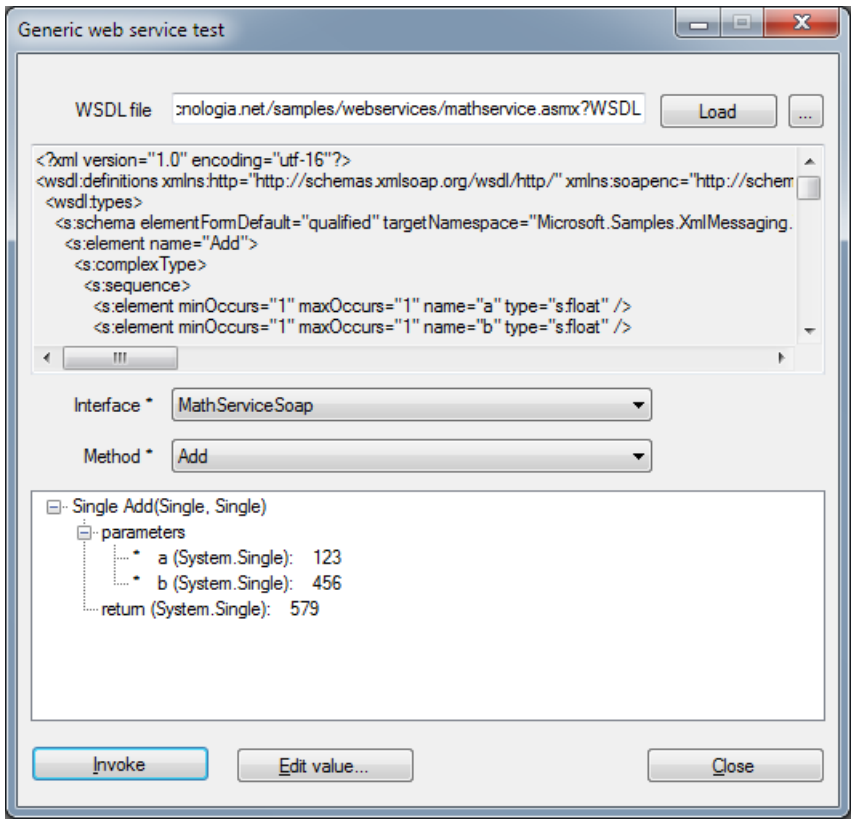
Once all required values have been supplied, then the SOAP operation may be executed by clicking the **Invoke** button on the test dialog (Figure 20 – Testing a SOAP operation). When execution completes successfully, the following dialog is displayed:

Figure 22 – Successful call notification



The returned values are presented in the test dialog. This may include output parameters and a return value.

Figure 23 – Test form following successful execution



If an error is encountered, a suitable error dialog is displayed.

4.3.4 GWSA Transaction Editor – Inputs tab

The main features of this tab are described in section 2.4.1.1.

4.3.5 GWSA Transaction Editor – Outputs tab

The main features of this tab are described in section 2.4.1.2.

4.4 Usage guidelines

- Do:
  - Use the Data Object Definitions dialog to help define Portrait data types that match the Web service data types.
    - This reduces the mappings that have to be configured
    - Simple mappings give better performance
    - The definitions may be edited later in the same way all custom Data Object definitions are configured.
    - The Data Object definitions may be reused for many methods and even services.
  - Validate parameter values
    - Since Portrait Foundation stores data in a reduced number of data types, for example, four-byte signed integer for all integral values (signed and unsigned), there is the possibility of overflow when converting to .Net types such as System.Int16, System.Uint32.
- Don't:
  - Use an operation that returns a large amount of data and only use a small portion, for example, retrieve all personal details when only a name or address is required. This causes
    - High network bandwidth requirements

- Potentially difficult mapping, for example, to extract each address element separately
- Potentially complex post processing and model iteration. For example, if an array of addresses is returned and only the home address is required, all addresses must be passed into the model context since mapping does not support array indexing.

4.5 Known limitations and issues

The GWSA is a re-usable component that provides a high degree of configurability and relative ease of use. This generic functionality must be traded-off against the more complex data processing that is possible when writing custom code for each SOAP operation.

A number of limitations are imposed by the underlying .Net Framework classes and Portrait Foundation itself imposes some additional limitations. Some limitations affect specific Web services only, whilst others are more general.

A number of known limitations are described below, along with some workarounds. It is possible that further limitations will be discovered as different Web services are encountered.

4.5.1 Only methods declared using the Http Protocol can be called

.Net Framework classes will only generate a proxy for the SOAP protocol over HTTP. Older XML Web services using GET and POST are not supported.

A workaround is to re-implement or wrap the Web service with one that uses SOAP over HTTP.

4.5.2 SOAP headers

Backwards-compatible mode

The original (ASP.NET compatible) GWSA had special handling for SOAP header values, with their own combo items in the input and output mapping dialogs. These are now only shown in 'backwards-compatible' mode (see section 4.6).

The Soap Header objects are derived from a framework class. The inherited properties of this class (such as Agent and MustUnderstand) are visible in the derived header's type definition.

The header supplies optional information and the Web service need not take this into account, but some Web services may interpret these values. Where the Web service ignores these values it is a modelling overhead to have to set the additional values.

Set to constant values rather than exposing these properties in models.

WCF and SOAP headers

The current (WCF compatible) GWSA does not recognise SOAP headers as such, but they may appear as normal input or output parameters, and can be mapped to Data Objects in the same way as other method arguments.

- inherited ASMX soap header properties such as 'MustUnderstand' are now not shown (these were displayed by the original GWSA).
- soap headers can only be handled in this way with ASP.NET 1.1 web services, because ASP.NET 2 generates 'anyAttribute' which we cannot handle.

WSE compatibility

No WSE support is provided by GWSA.

4.5.3 Portrait asynchronous SOAP transactions are not supported

The architecture of the Generic Web Service Adapter is .Net based, using a compiled proxy. It is not possible to implement an adapter with the same 'Wait for inbound SOAP message' semantics.

The workaround is to use Portrait External Event Nodes to create a correlation ID, the Generic Web Service Adapter to send the outbound SOAP request, Portrait Foundation Web Services to receive inbound SOAP responses and the correlation ID to notify the External Event.

4.5.4 Arrays (Data Object Collections) must be mapped in their entirety

There may be a desire to only extract part of a collection in a SOAP response; however the Configuration Suite mapping dialog does not allow collections to be indexed.

Other Portrait Foundation nodes may be used to post-process the returned data, for example, script node or collection iteration but it may be expensive to pass a complete collection to the Model context only to access a single element. Alternatively a custom DAT can be written.

Note that the GWSA has been written to cater for indexing into arrays during mapping and this would be available if the functionality is implemented in the Configuration Suite mapping dialog.

4.5.5 WCF compatibility

Technical design decisions, made to limit the range of WCF issues which we need to handle, include:

- Use XMLSerializer not WCF's default DataContractSerializer

  System.Xml.Serialization can handle a much broader range of XML Schema constructs than System.Runtime.Serialization.

  Furthermore, the DataContractSerializer generates System.Runtime.Serialization.ExtensionDataObject; but the GWSA cannot consume most System types (see above, 4.1.1 'Supported data types').

- Support wsHttpBinding and basicHttpBinding only

  wsHttpBinding will consume most WCF services and basicHttpBinding will consume legacy ASP.NET web services.

- No support for WCF Fault Contracts

  WCF will raise a FaultException but GWSA has only generic handling of exceptions.

4.5.6 "Specified" properties

Many of the types that are created by the proxy generator include properties with names that are appended with "Specified". These properties are used along with the property of the same name that does not include the "Specified" suffix.

We recommend that such a property, when an Input, should be mapped to a Boolean constant with value 'true', and when an Output or return value, should be ignored. The extra properties will then have no effect outside the GWSA DAT.

4.6 Backwards compatibility

WCF is Microsoft's latest technology within the Service Oriented Architecture (SOA) family, which exposes server functionality through a wide range of services. WCF is compatible with the previous ASP.NET web services, and is therefore our only GWSA solution going forward.

However we do not wish to force existing clients to reconfigure existing GWSA clients. Therefore we support the old technical solution for:

Design time (GWSA Config)

- View existing GWSA systems and transactions.
- Add a new GWSA transaction using the old code to an existing ASP.NET compatible GWSA system.
- If a new transaction is configured within an existing ASP.NET compatible GWSA system, the Portrait Data Objects generated must be compatible with existing GWSA Data Objects.

Runtime (GWSA Adapter)

- For old GWSA configuration: use old code for WSDL import, proxy generation and invoke web service method

All new GWSA systems will use the new WCF framework.

The 'New generic web service system', 'New web service transaction' and GWSA DAT 'Properties' menu items will automatically uphold these requirements.

In the GWSA Transaction Editor, you can tell at a glance which version is in use from the Command tab: the upper drop-down is labelled 'Service' for the ASP.NET compatible version, and 'Interface' for the new WCF compatible version.

Appendix A OLE DB data source testing summary

Table 5 – Summary of **OLE DB data source testing**

Source	Version	Provider	Comments
Oracle	8i	Oracle Provider for OLE DB (version 8.17.3.0) MDAC 2.8	Extensive testing SQL statements, stored procedures and functions all execute successfully. National language support has not been tested.
Oracle	8i	Microsoft OLE DB Provider for Oracle MDAC 2.8	Minimal testing The GDA components are unable to retrieve record sets from stored procedures (the provider does not support REF CURSOR parameters using native OLE DB .Net Framework classes).
SQL Server		Microsoft OLE DB Provider for SQL Server MDAC 2.8	Extensive testing SQL statements, stored procedures and functions all execute successfully. National language support has not been tested.
SQL Server		Microsoft OLE DB Provider for ODBC	Connectivity testing The support for this provider used with a SQL Sever is currently not supported by the Portrait Generic Access. It is recommended to use the Microsoft OLE DB Provider for SQL Server in stead. This provider has better performance characteristics.
Excel		Microsoft Jet 4.0 OLE DB Provider MDAC 2.8	Connectivity testing Simple SQL statements executed successfully.
SQL Server		SQL Server Native Client 11.0	Tested with SQL Server 2014

Appendix B Portrait scalar data types

The following table contains descriptions of the Portrait scalar data types. It shows the **Enum** item name (the name under which the type is shown in the Configuration Suite) and the .NET Framework type (the type used within .NET software to represent this Portrait data type).

Table 6 – Portrait scalar data types

Enum item name	Description	.NET Framework type
Integer	A variant of type VT_I4 (a four-byte signed integer value).	Int32
String	A variant of type VT_BSTR (a string obtained and freed by the BSTR functions).	String
FloatingPointNumber	A variant of type VT_R8 (an eight-byte IEEE real value).	Double
DateTime	A variant of type VT_DATE (a value denoting a date and time, represented as a double-precision number, where midnight, January 1, 1900 is 2.0, January 2, 1900 is 3.0, and so on. This is the same numbering system used by most spreadsheet programs, although some specify incorrectly that February 29, 1900 existed, and thus set January 1, 1900 to 1.0.).	DateTime
Boolean	A variant of type VT_BOOL (a 16 bit Boolean value, where a value of 0xFFFF indicates True and a value of 0 indicates False)..	Boolean

Appendix C OLE DB data types

The following table contains descriptions of the OLE DB data types. It shows the **Enum** item name (the name under which the type is shown in the Configuration Suite), the OLE DB data type (shown in parentheses) and the .NET Framework type (the type used within .NET software to represent this OLE DB data type).

Table 7 – OLE DB data types

Enum item name	Description	.NET Framework type
BigInt	A 64-bit signed integer (DBTYPE_I8).	Int64
Binary	A stream of binary data (DBTYPE_BYTES).	Array of type Byte
Boolean	A Boolean value (DBTYPE_BOOL).	Boolean
BSTR	A null-terminated character string of Unicode characters (DBTYPE_BSTR).	String
Char	A character string (DBTYPE_STR).	String
Currency	A currency value ranging from -2^{63} (or -922,337,203,685,477.5808) to 2^{63} -1 (or +922,337,203,685,477.5807) with an accuracy to a ten-thousandth of a currency unit (DBTYPE_CY).	Decimal
Date	Date data, stored as a double (DBTYPE_DATE). The whole portion is the number of days since December 30, 1899, while the fractional portion is a fraction of a day.	DateTime
DBDate	Date data in the format yyyymmdd (DBTYPE_DBDATE).	DateTime
DBTime	Time data in the format hhmmss (DBTYPE_DBTIME).	TimeSpan
DBTimeStamp	Date and time data in the format yyyymmddhhmmss (DBTYPE_DBTIMESTAMP).	DateTime
Decimal	A fixed precision and scale numeric value between -10^{38} -1 and 10^{38} -1 (DBTYPE_DECIMAL).	Decimal
Double	A floating point number within the range of -1.79E +308 through 1.79E +308 (DBTYPE_R8).	Double
Empty	No value (DBTYPE_EMPTY).	(Not applicable)
Error	A 32-bit error code (DBTYPE_ERROR).	Exception
Filetime	A 64-bit unsigned integer representing the number of 100-nanosecond intervals since January 1, 1601 (DBTYPE_FILETIME).	DateTime
Guid	A globally unique identifier (or GUID) (DBTYPE_GUID).	Guid
IDispatch	A pointer to an Idispatch interface (DBTYPE_IDISPATCH).	Object
Integer	A 32-bit signed integer (DBTYPE_I4).	Int32
Iunknown	A pointer to an Iunknown interface (DBTYPE_UNKNOWN).	Object
Numeric	An exact numeric value with a fixed precision and scale (DBTYPE_NUMERIC).	Decimal
PropVariant	An automation PROPVARIANT (DBTYPE_PROP_VARIANT).	Object
Single	A floating point number within the range of -3.40E +38 through 3.40E +38 (DBTYPE_R4).	Single
SmallInt	A 16-bit signed integer (DBTYPE_I2).	Int16

Table 7 – OLE DB data types

Enum item name	Description	.NET Framework type
TinyInt	A 8-bit signed integer (DBTYPE_I1).	Sbyte
UnsignedBigInt	A 64-bit unsigned integer (DBTYPE_UI8).	Uint64
UnsignedInt	A 32-bit unsigned integer (DBTYPE_UI4).	Uint32
UnsignedSmallInt	A 16-bit unsigned integer (DBTYPE_UI2).	Uint16
UnsignedTinyInt	A 8-bit unsigned integer (DBTYPE_UI1).	Byte
Variant	A special data type that can contain numeric, string, binary, or date data, as well as the special values Empty and Null (DBTYPE_VARIANT).	Object
Wchar	A null-terminated stream of Unicode characters (DBTYPE_WSTR).	String