mv.NET Solution Objects

Developer Guide

A product from BlueFinity

Copyright Notices

Contacting Us

We are always very happy to be able to discuss all aspects of our products with our customers - prospective and current alike. You can contact us via the following means:

Website:	www.bluefinity.com
Email:	support@bluefinity.com
Address:	10260 SW Greenburg Road, Suite 400, Portland, OR 97223, USA
Address:	575-599 Maxted Road, Hemel Hempstead, Herts, HP2 7DX, UK

Trademark Acknowledgements

Contents

Welcome to mv.NET

Firstly, thank you for either purchasing one or more of the mv.NET products, or for taking the time to explore the great functionality that they can provide to you and your fellow developers.

This chapter outlines the members of the mv.NET family of products and also summarizes the contents of this guide.

The mv.NET Family of Products

Solution Objects is one of the members of the mv.NET family of products authored by BlueFinity. mv.NET is ***the*** essential tool for any MultiValue database developer wishing to create .NET based application interfaces to their current or new MultiValue database file system.

The design goal of mv.NET is to enable the MultiValue developer to combine the power and flexibility of proven MultiValue technology with the state-of-the art, feature rich .NET environment. Its design also enables and encourages the developer to leverage, wherever possible, previously acquired MultiValue skills.

BlueFinity's team of software engineers has huge knowledge and experience of using both MultiValue systems and the .NET environment. We proudly regard ourselves as being one of the foremost companies in providing this technology bridge and look forward to working with you to enable you to meet your software development goals.

Feature Overview

The Solution Objects product provides the ability to create a strongly-typed class-based access layer to your MultiValue database.

The product's key features are as follows:

- Entity modeling tool to allow database-to-entity mapping definitions to be created
- Schema import wizard to allow quick creation of entity models
- Code generation utility to automatically create .NET code modules in either C# or VB.NET
- Run-time support assemblies for generated code
- Full support for nested data down to sub-value level

The mv.NET Suite

Solution Objects is one of three products within the mv.NET suite; the suite as a whole comprising of:

- **Core Objects** - object oriented native .NET access to MultiValue databases.
- **Solution Objects** - Strongly-typed class-based access to your MultiValue database.
- **Adapter Objects** - complete implementation of an ADO.NET managed data provider for MultiValue databases, offering a standardized interface to database access.

Developer Guide Contents

The contents of this guide are designed to provide a basis for learning about the Solution Objects module. Further help is provided within the Visual Studio environment using the product's dynamic and IntelliSense help systems.

Solution Objects Overview and Getting Started

Accessing your database information is one of the most important aspects of creating line-of-business applications. For .NET developers wishing to access MultiValue databases, mv.NET has provided strong connectivity capabilities for a number of years with its Core Objects and Adapter Objects component sets.

Solution Objects adds to this connectivity capability by introducing the ability to easily create a strongly-typed, class-based access layer to your MultiValue database. This access layer brings a number of significant benefits for the application developer:

- Simplified, intuitive access to the underlying database, without the need for specific MultiValue database knowledge
- Compile-time data-type checking of application code
- Support for native .NET databinding for both Web and WinForm applications

Solution Objects - Basic Concepts

Before diving into too much detail, it would perhaps be useful to set the scene in terms of the basic concepts that Solution Objects brings to the table. The sections below cover the main ideas that you'll need to grasp before starting to use Solution Objects.

Entities

First of all, we have the concept of "Entities". An entity is a 'thing' that your application deals with and, as such, can be a representation of pretty much anything - something physical or something abstract - it all depends on the "domain" that your application deals with.

If your application is a stock control system chances are that your entities will be things like Products, Suppliers, Purchase Orders etc. If you are creating a banking application your entities will be things like Bank Accounts, Customers, etc.

The modern object-oriented programming paradigm fits pretty well with the concept of entities because, very often, many of the "Objects" inside your application are a representation of the entities within your "application domain".

If you have an existing application, your database structure will probably be the first representation that you can turn to in order to identify the entities that exist within your application domain. Sometimes there will be a strikingly close match between files and entities - sometimes not. It all depends, of course, on the mindset of the person/people who designed the file structure originally.

Anyway, the first thing that you need to establish is a good understanding of what entities it makes sense for your application to use and then combine this with a good understanding of where the data that these entities use is held within your database.

You also need to understand how entities relate to one another. This, again, may well be represented to a certain extent by the dictionary definitions within your database.

Entity Models

An Entity Model is a formal definition of all the entities that exist within your application domain. It also contains information describing how entities relate to each other. Solution Objects also uses the Entity Model to contain the following additional pieces of important definition information, including:

- how entities map onto the underlying data store (MultiValue database)
- how entities can be selected
- how back-end database resident routines are to be accessed

You are able to create any number of different entity models as required.

Data and Business Access Layers

.NET developers live and breathe classes; after all, the .NET framework is the mother of all class collections! Thus, it is natural for this community of developers, when dealing with the matter of application data, to view their application data as being made up of a series of classes.

Therefore, the basic idea behind Solution Objects is to assist you in the task of producing a series of class definitions that represent your application's main data environment. These classes being able to read and write data from/to a MultiValue database as required by the logic of your application.

A Data Access Layer (DAL) is a series of classes - each referred to as a Data Access Class (DAC) - which provides the "first line" of class-based abstraction of your database. Each DAC knows exactly where to go within the associated database in order to retrieve the persisted data that it needs/represents.

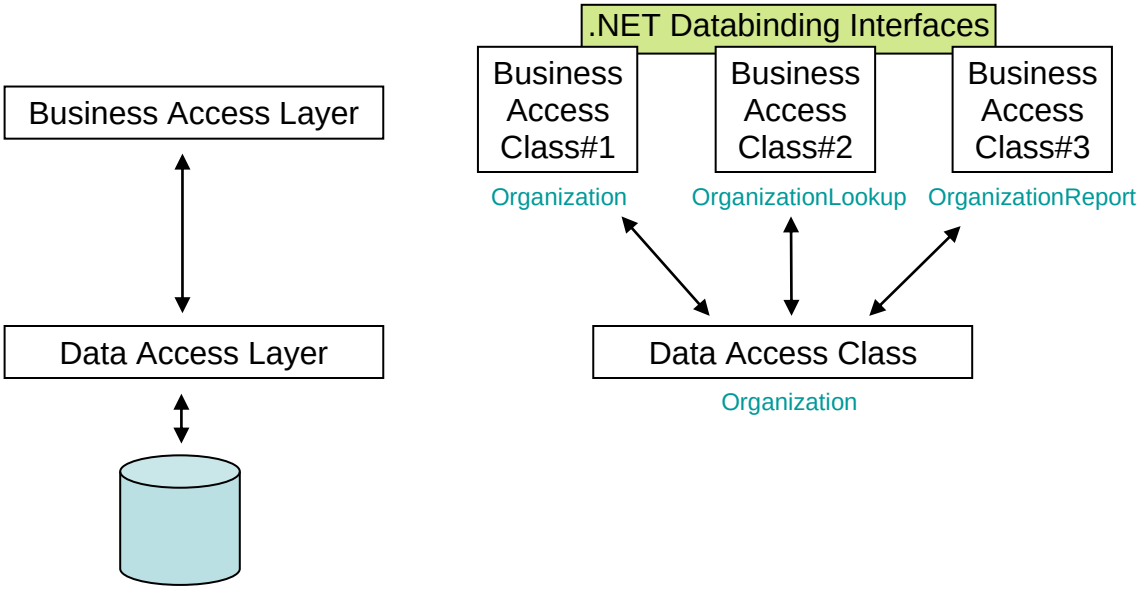
A Business Access Layer (BAL) is also a series of classes - each referred to as a Business Access Class (BAC) - with which your application code interacts. Some people use the term Business Logic Layer (BLL) as the name of such a concept. The BAL has no knowledge of where persisted data comes from - it lets the DAL handle that. The classes within the BAL, of course, have a very strong association with the classes in the DAL, but these classes may have different names and a single DAC may be represented by multiple BACs.

The BAL is more concerned with providing a series of classes that are useful for the application developer. These classes, for example, know how to interact with data binding mechanisms; they can also present a subset of the classes (and

properties within the DAL classes) in order to simplify the class interface or to prevent the application developer from accessing sensitive, special or volumous data.

The end application developer does not interact directly with the DAL. The DAL is there to service the needs of the BAL.

The following diagram illustrates these concepts:



Solution Objects - Process Summary

There are a number of distinct tasks that you will need to perform in order to produce a functional class-based access layer using Solution Objects:

1. Install the mv.NET Client Interface Developer SDK. The mv.NET Data Manager utility is installed as part of this SDK and is used to perform tasks 2 through 6 below.
2. Download the Entity Models Repository files into a designated account on your MultiValue database.
3. Create an empty Entity Model.
4. Use the entity generator to create your initial entity definitions based on existing database dictionary/schema definitions.
5. Flesh out your entity definitions as required, including the definition of one or more Business Access Layers.

6. Use the code generator to generate your Data Access Layer (DAL) and Business Access Layer (BAL) code modules.
7. Include these code modules inside a Visual Studio project and build your access layer assembly. You can have the code modules in separate Visual Studio projects if you prefer to produce separate DAL and BAL assemblies.
8. Reference your BAL assembly from within your end application Visual Studio solution.

The following sections cover each of the above steps in turn.

Initial Installation

Solution Objects is supplied as part of the mv.NET Client Interface Developer (CID) SDK package. Therefore, the first step is to load this onto your development workstation. If you already have a pre-4.1 version of the mv.NET CID installed on your system you may upgrade it using the latest CID service pack. Please refer to the Core Objects developer guide and the current release notes for more information on upgrading an existing CID installation.

Entity Models Repository Download

The Data Manager stores all entity model definitions in a series of files within a MultiValue database account. Collectively, these files are referred to as the "Entity Models Repository" (EMR). The decision as to which account is to host the EMR is ultimately your decision, but we recommend that you create an account specifically for this purpose and call it "MV.EMR". **Please do not use the MV.NET account that has been created to host the mv.NET server-side components as the EMR host account.**

Once you have created the EMR account or identified an existing account that you are going to use for this purpose, you need to make sure that this account has been "enabled" for mv.NET usage. Please refer to the Core Objects Developer Guide for further details on this topic.

Once the EMR account has been enabled, you need to create a server and account profile within the Data Manager (if ones do not already exist) to connect into this account. **Note, if you have created a dedicated account for your EMR you can turn**

'File schema caching' on within the account profile ('Other' tab) in order to improve performance of the entity definition aspects of the Data Manager.

Once you have your server and account profiles in place, you then need to create a Login profile within the Data Manager to reference this server/account profile pairing. Again, please refer to the Core Objects Developer Guide for an in-depth explanation of this process.

Next, you need to use the Data Manager's Server Console window to create the EMR files and schema details. You will have already used the Server Console window to enable the EMR account - this time, once you are at command level within the EMR account within the Server Console window, you need to select the "Download Entity Models Repository" option from its top "Action" menu.

This option takes about 2 or 3 minutes to complete. When it is finished, 12 files will have been created within your database account all with a name starting with "SO_". The dictionary items for these files will also have been downloaded.

The final step in this phase is to tell the Data Manager which login profile it is to use to access the EMR. This is defined by right-clicking the "Entity Models" node at the bottom of the Data Manager's treeview area and selecting the "Set Entity Models Repository Location" option from the resulting popup context menu. The resulting dialog window then allows you to select the relevant login profile name.

Once these steps have been performed, you will be ready to start creating entity model definitions.

Creating a New Entity Model

The first thing that you need to do after establishing the EMR is to create the first entity model. You can do this by right-clicking the "Entity Models" node within the Data Manager's treeview area and selecting the "Set Entity Models Repository Location" option from the resulting popup context menu. Please refer to the Maintaining Entity Model Definitions chapter for more details on this process.

Generating Initial Entity Definitions

After creating your first entity model entry, you will need to start creating entity definitions. This can be done one at a time manually or it can be done using the entity definition generator. The generator scans the schema of the database files

that you specify and infers from this information the entities (and properties within these entities) that should be created. Please refer to the Creating Entity Definitions chapter for more details on this topic.

Extending Entity Definitions

Once the initial details of an entity have been created either manually or via the entity generator, you will need to spend some time fine tuning and extending these details. You will first need to make sure that the Data Access Class (DAC) definition contains all of the required member details (properties, selection methods and subroutine methods). Then you can start creating one or more Business Access Class (BAC) definitions as required by the end-application developers. Finally, you will need to create one or more Business Access Layers (BAL) to gather together the required BACs into a single access layer. Please refer to the Maintaining Data Access Class Definitions, Maintaining Business Access Class Definitions and Maintaining Business Access Layer Definitions chapters for more information on these topics.

Generating DAL and BAL Code

One you have created the definitions of your DACs, BACs and BALs you are ready to generate some code. Please refer to the Generating Code Modules chapter for details on how you can do this.

Using Generated Code

All of the previous actions are performed using mv.NET's Data Manager utility. The next step involves firing up Visual Studio and creating a new Class Library project to host your generated code. Please refer to the Using Generated Code chapter for details on how to do this.

Referencing and Using Your Business Access Layer

Finally, in order to utilize the fruits of all your hard labors, within your end application project you simply need to reference the BAL assembly that you have produced using Visual Studio in the previous step. Please refer to the Utilizing a Business Access Layer chapter for further details on this topic.

Sample Projects

Some sample projects illustrating the use of a generated BAL are installed as part of the CID installation. These samples can be found in the following folder:

C:\Documents and Settings\All Users\Application Data\BlueFinity\mv.NET\Version4.0\Examples\Solution Objects

Note, on Vista/Server 2008 systems **C:\Documents and Settings\All Users\Application Data** is represented as **C:\ProgramData**

Maintaining Entity Model Definitions

This chapter describes how you are able to create and maintain entity model definitions.

Overview

An entity model definition contains everything that Solution Objects requires in order to generate the code to support a class-based representation of an application's data domain.

As such, it contains a range of detailed information. This information and the creation and maintenance of such information are described in the following chapters of this guide. However, there are a number of pieces of information that describe the very top-level characteristics of an entity model and it is this information that you associate directly with the entity name itself.

The Entity Models Node

The second to last main treeview node within the Data Manager contains all of the entity modeling functionality. When expanded, this node will contain either a list of entity model repository locations or the entity models within a single repository.

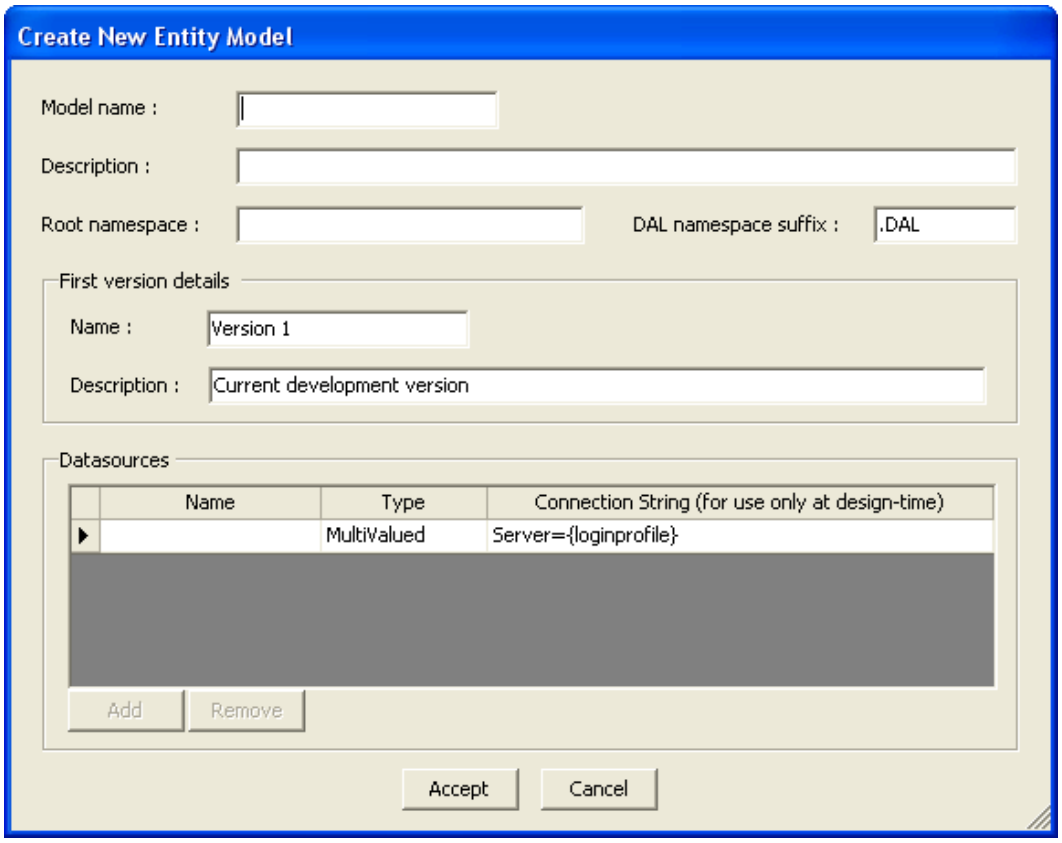
By default, only a single entity model repository at a time is displayed within the Data Manager. This can be changed to allow multiple repositories to be accessible at the same time by selecting the "Use Multiple Repository Locations" option from

the right-click menu of the "Entity Models" node. On selecting this option, the node caption changes to "Entity Model Repositories" and the content of the right-click menu changes to reflect the shift in functionality. To return back to a single repository at a time behavior select the "Use a Single Repository Location" menu option.

When the multiple repository location option is active you may create many "location entries" within the "Entity Model Repositories" node. Each location entry is simply a logical name associated with a login profile name – the login profile name being the one that connects into the relevant EMR.

Creating an Entity Model Definition

You can create a new entity model definition by right-clicking the "Entity Models" node within the Data Manager's treeview area and selecting the "Create New Entity Model" option from the resulting popup context menu. If the multiple repository locations option is active the "Create New Entity Model" option is available from the right-click context menu of each repository location node. On selecting the create option, the following dialog window is displayed:



The above window allows you to enter the initial pieces of information about this entity model. The sections below describe each input field/area.

Model name: The name of the model which will be used as the entry within the Data Manager's treeview area and also within program code. It therefore only allows certain characters to be used in the name.

Description: A description of the entity model's purpose or content.

Root namespace: The namespace to be used within all generated code. All namespaces typically start with the relevant owning organization's name, followed by an indication of the purpose of the namespace area. For example:

```
BlueFinity.SOP
```

If you are unsure, please refer to .NET programming documentation for an explanation of the purpose and format of namespaces.

DAL namespace suffix: The characters to be appended to the end of the root namespace in order to identify/group together the classes comprising the DAL.

First version details: The next section of the input window then allows you to enter a name and description for the first version of the entity model which will be created as an implicit part of the creation of this new entity model.

Datasources: The final section of the input window allows you to enter the details of the place where application data is to be found. The current version of Solution Objects only supports a single datasource, which must be a MultiValue database. Therefore, this screen allows you to amend 2 pieces of datasource information:

Datasource name: The name/identifier of the datasource to be used in program code.

Datasource connection string: The connection string to be used by the Data Manager to connect into the datasource. The connection string must be of the format:

```
Server={login profile}
```

Where *{login profile}* represents the name of a login profile defined within the Data Manager.

Clicking the Accept button will store the initial definition details for the new entity model and the first version within the EMR.

Modifying an Existing Entity Model Definition

Once you have created an entity model, you are able to view and amend its top-level definition information by right-clicking the appropriate entity model node within the Data Manager's treeview area and selecting the "Maintain Model Details" option from the resulting popup context menu.

Maintaining Entity Model Versions

This chapter describes how you are able to create and maintain different versions of your entity model definitions.

Overview

It is very likely that once you have generated and used your first entity model in an application that has gone into production, you will need to make sure that the entity model definition information is not changed other than to fix issues that occur whilst it is in production. Therefore, if you need to onward develop the entity model for use in the next version of an application or applications, you will need to create a new version of the entity model to contain the enhancements. For this reason, Solution Objects has the concept of creating multiple versions of an entity model.

Creating a New Entity Model Version

In order to create a new entity model version, right-click "Versions" node beneath the appropriate entity model node within the Data Manager's treeview area and select the "Create New Version" option from the resulting popup context menu. .

Creating Entity Definitions

This chapter describes how you are able to create the initial definition of one or more entities.

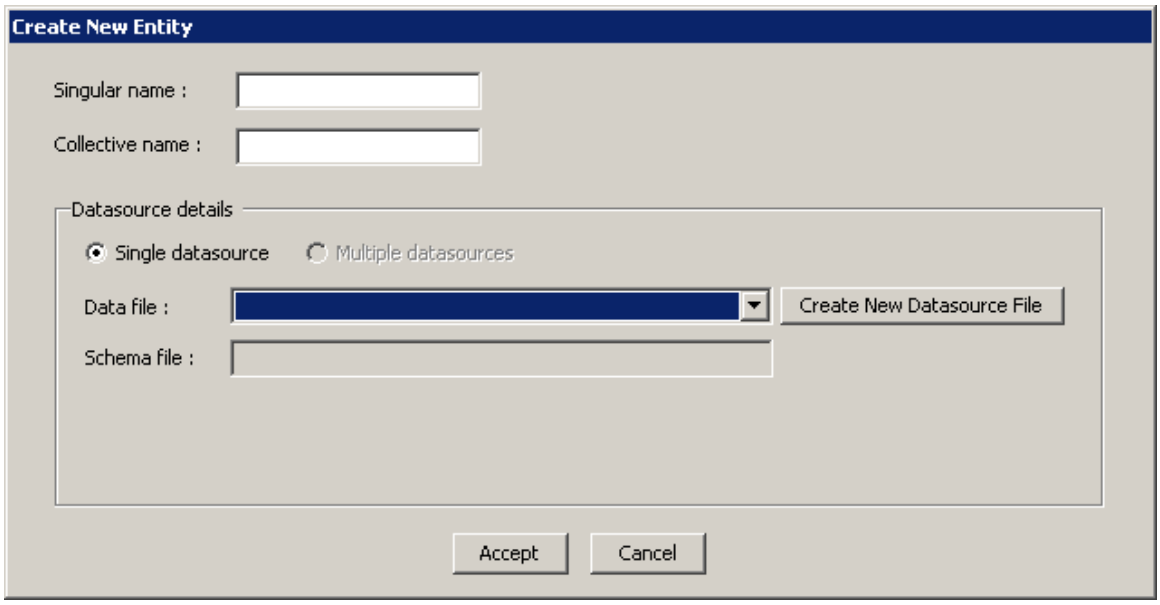
Overview

The building blocks of an entity model are entity definitions. Thus, clearly, one of the main things that the entity model area of the Data Manager needs to do is to allow you to create (and then onward maintain) entity definitions.

Creating Entity Definitions Manually

The first way of creating a new entity definition is to select the "Create New Entity" option from the right-click context menu of the "Entities" node. This node is positioned directly beneath the relevant version node within the Data Manager's treeview area.

When this is done, the following dialog window will be displayed:



Firstly, this window allows you to enter the 2 versions of the entity's name. The singular name is used to refer to a single instance of the entity. The Collective name is used to refer to a collection of instances of the entity. This dialog is the only place where you are able to define these 2 names.

The next area of this window allows you to select the base file of the entity. This file will provide the schema information that will be used when you create the datasource mapping aspect of the Data Access Class definition (see next chapter).

Clicking the Accept button will create the initial entity definition and a new node will be created underneath the Entities node representing the presence of the new entity.

The "Create New Datasource File" button allows you to create a new file to link the new entity to.

Notes on Manual Entity Creation

There are a few notes worth mentioning on the topic of creating entities manually.

Firstly, if your entity's property interface is going to closely match its base file's schema, it will normally be much quicker to use entity generating (see next section) as opposed to using the manual approach.

Secondly, the list of files presented in the Data file combobox on the above form will only display the names of files which are not already represented by an entity definition.

Thirdly, if you are creating an entity for a file that contains nested data, you must (in fact, you are forced) to create the entity that represents the single-valued fields first. Once this top-level entity has been created you are then able to create entities which represent the nested data elements of the file. These nested elements will be listed within the Data file combobox using one of the following formats:

{FileName} {MvGroupName} (MvGroup)

Or (for sub-valued nested data)

{FileName} {SvGroupName} (SvGroup)

Where *{FileName}* is the base file's name, *{MvGroupName}* is the name of the Mv group as defined in the extended dictionary of the file and *{SvGroupName}* is the name of the Sv group as defined in the extended dictionary of the file.

Generating Entity Definitions

An alternative way of creating entities is to get Solution Objects to do the initial heavy work for you. It does this by scanning the schema information of one or more files (you choose which ones it scans) and then suggests an entity structure of one or more entities based on its findings.

A pre-requisite of driving entity generation from your file schema is that you spend some time making sure that the Extended Dictionary information for each file has been created. Extended Dictionary maintenance is covered in depth with the CoreObjects Developer Guide, but as a quick heads-up on what you need to focus on, below is a list of the Extended Dictionary sections that are of particular interest to the entity generator and, for that matter, Solution Objects in general:

Extended Dictionary Fields

You can maintain the extended dictionary information of a file by right-clicking the relevant file node within the Data Manager and choosing the 'Maintain File Schema' option from the resulting context menu. You can also maintain schema information from within the Data Access Class maintenance window.

Within the resulting window you need to make sure that the following fields (in the 'Extended' tab in the lower section of the window) are set as required:

- Data type,
- MV type
- MV group

- SV group
- .NET column name (this is used as the default property name)
- Input Criteria (tab) fields
- Dependencies (tab) fields
- File Link (tab) fields

File Properties

You can maintain the general properties of a file by right-clicking the relevant file node within the Data Manager and choosing the 'Properties' option from the resulting context menu.

Within the resulting window you need to make sure that the following fields (in the 'Extended' tab in the lower section of the window) are set as required:

- Auto item ID generation (group box fields)
- .NET Environment Name Mappings (group box fields)

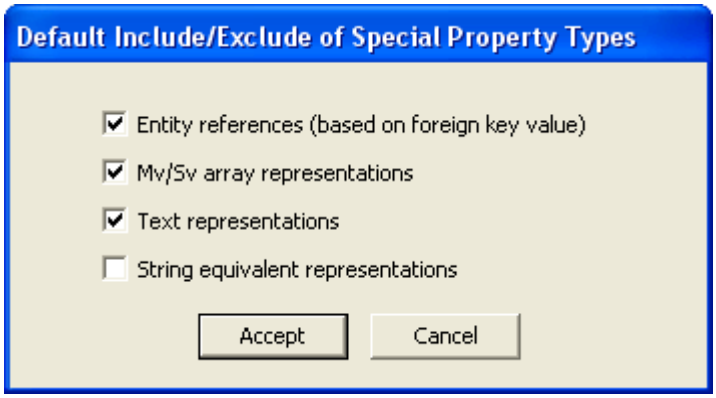
Within the .NET Environment Name Mappings fields, the 'DataTable/Entity name' field is used as the default singular name of an associated entity. The group name grid entries are used for the default singular names of entities representing nested data within the file structure.

For more information on all fields within the 2 above maintenance windows please refer to the Core Objects Developer Guide which is shipped as part of the mv.NET CID product.

Running the Entity Generator

Entity generation can be invoked by selecting the "Generate Entities From File Schema" option from the right-click context menu of the "Entities" node. This node is positioned directly beneath the relevant version node within the Data Manager's treeview area.

When this is done, the following initial dialog window will be displayed:



This dialog allows you to set the defaults for the generation of various "special" property types. These special property types are explained in detail within the following Maintaining Data Access Class Definitions chapter, but to summarize:

Entity references - these are properties that hold a reference to an instance of a related entity based on the presence of an item ID (foreign key) within the file.

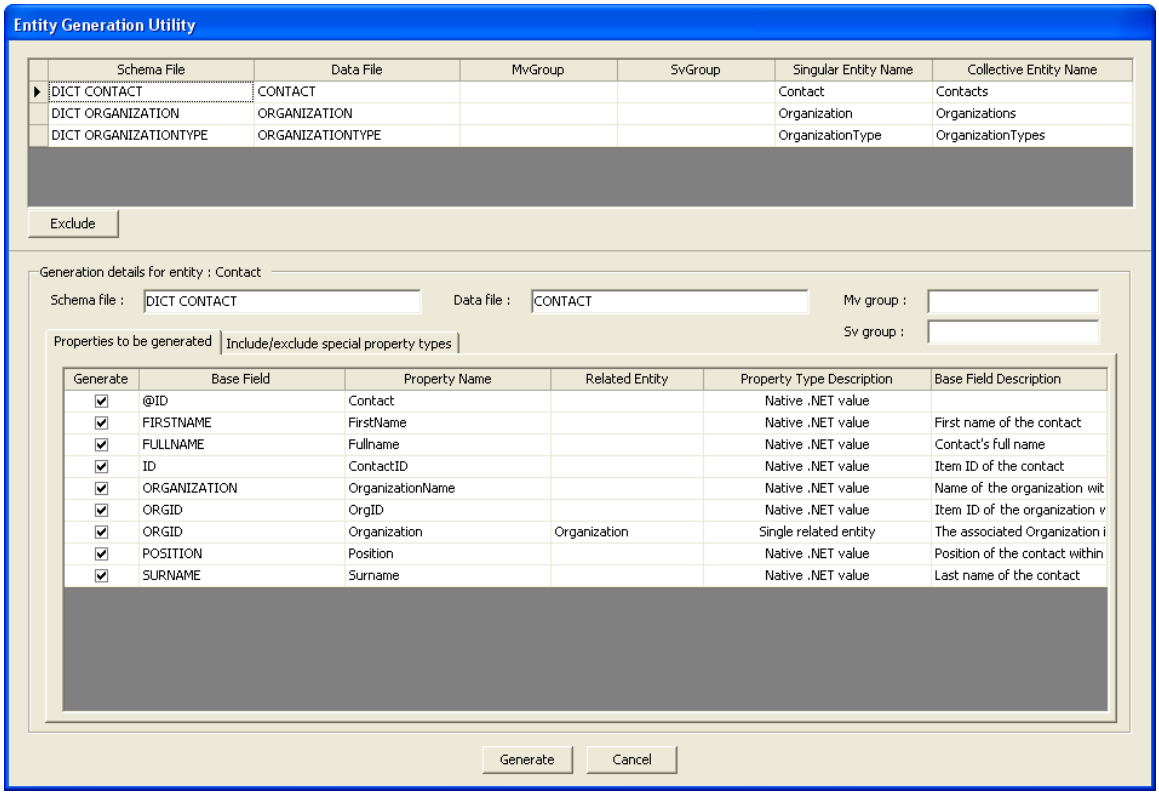
Mv/Sv arrays representations - these are properties that hold a reference to an instance of a specialized class (provided by Solution Objects) that represents a list of multi/sub/multi-sub values, based on the presence of nested data within the file.

Text representations - these are properties that hold a string value representing a list of multi/sub/multi-sub values, based on the presence of nested data within the file. The string value has the internal value delimiters replaced with alternative characters (e.g. CRLF) as defined by the DAC definition.

String equivalent representations - these are properties holding a string value representation of either a date or time field, as opposed to a value of .NET data type DateTime.

Note, these special values are in addition to the automatically included "simple" values representing normal base field data, numbers, strings, dates etc.

Once you have selected the required special property defaults, clicking the Accept button will result in the display of a dialog window which allows you to select those files which are to be scanned as part of the entity generating process. Select the files you require from the list and click the Accept button. The following main window will then be displayed:



This window allows you to review the results of the entity generator's scan and to edit various pieces of the data that will be used in the final definition generating phase.

The top section of the window lists the entities that will be generated along with the base file/schema that will be associated with the entity. You are able to edit the singular and collective names of the entities as necessary. If the scanning process has suggested some entities that you do not wish to generate, you can use the Exclude button to remove them from the list.

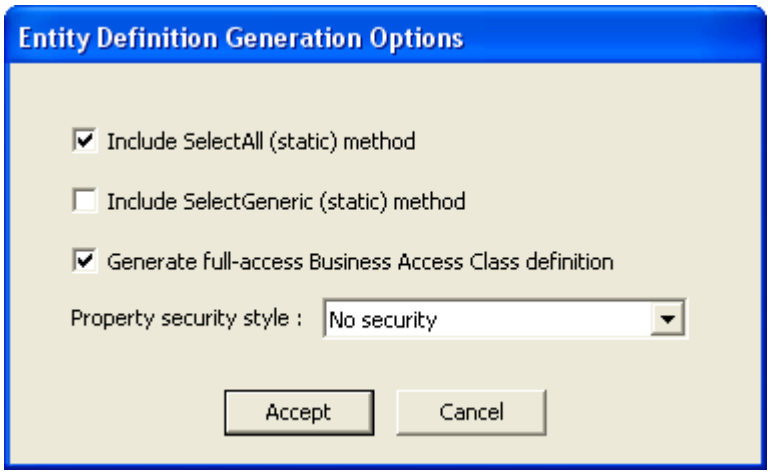
The lower half of the window lists the property definitions that are to be generated for the currently highlighted entity. You may control the generation of each property by checking/unchecking the Generate column as necessary. You may also edit the values in the Property Name, Related Entity and Base Field Description columns as necessary. You are able to highlight multiple rows within the property grid and alter each highlighted row's "Generate" column tick box setting en masse by using the right-click context menu of the grid.

The "Include/exclude special property types" tab simply allows you to adjust the settings entered within the initial special properties dialog window on a per-entity basis. If you adjust any settings within this tab, the Generate column checkbox values for that entity will be adjusted accordingly.

It is important that you review the information presented in this window carefully.

When you are OK with the settings, clicking the Accept button will force the generator to validate the generation details. The validation process will check that all entity names and property names within an entity are unique. If any problems are found, a list of errors will be displayed. This list of errors may be shown/hidden by use of the " View/Hide Generation Errors" button in the bottom left of the window.

If no validation errors are detected, the following dialog will be displayed:



This dialog allows you to specify various settings that will be applied to all generated entities. Note, these settings may be adjusted later on a per-entity basis.

These entity settings are explained in detail within the following Maintaining Data Access Class Definitions chapter, but to summarize:

Include SelectAll - this indicates whether each DAC is to have a selection method named "SelectAll". This method allows the entire content of the base file to be selected.

Include SelectGeneric - this indicates whether each DAC is to have a selection method named "SelectGeneric". This method allows the content of the base file to be selected/sorted based on datasource specific select/sort command syntax.

Generate full-access BAC - this indicates whether a BAC with the same name as the entity is to be generated in addition to the DAC. This BAC will contain all properties and methods of the base DAC.

Property security style - this indicates what style of property-level security will be applied to each entity.

Clicking the Accept button will create the initial entity definitions and new nodes will be created underneath the Entities node representing the presence of the new entities.

Maintaining Data Access Class Definitions

This chapter describes how you are able to maintain the definition of the Data Access Class associated with an Entity .

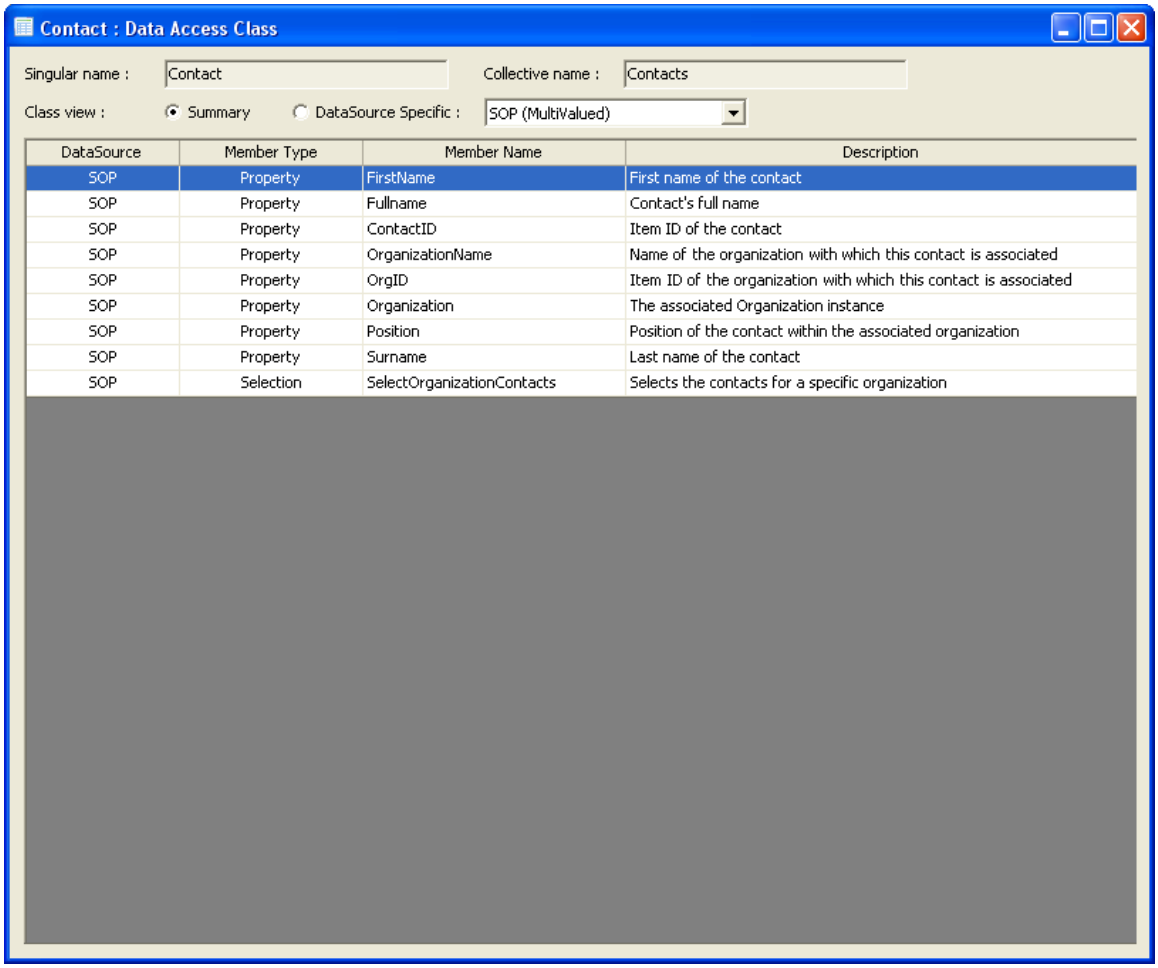
Overview

At the heart of each entity definition is a description of the data that the entity is to access and utilize from the underlying datasource. This information is contained with a Data Access Class (DAC) definition. Each entity, thus, has one (and only one) associated DAC definition.

Each entity node within the Data Manager's treeview has a sub-node named "Data Access Class". Double-clicking this node takes you into the DAC maintenance window.

DAC Summary Display

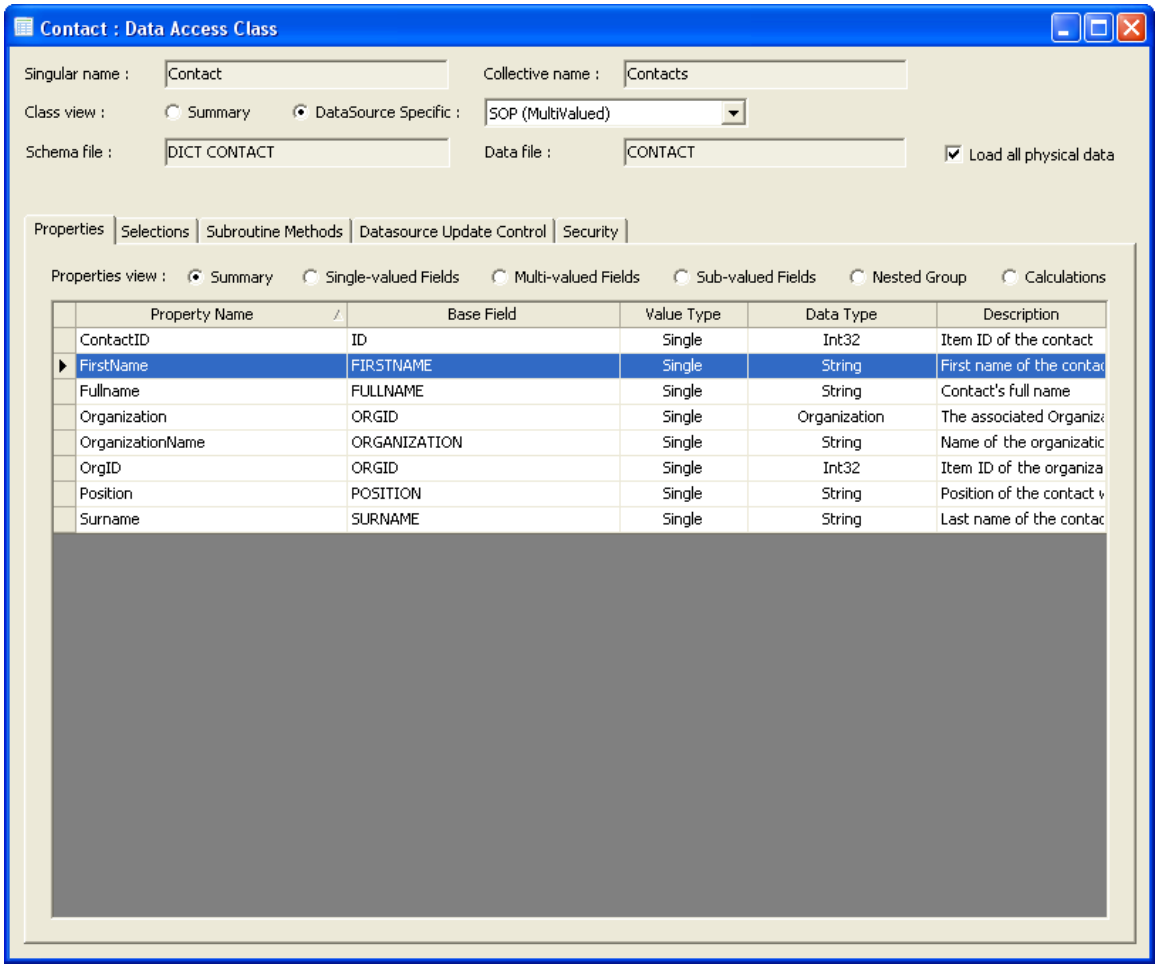
When the DAC maintenance screen is first displayed, you are shown a summary of all members on the class. In the current version on the product, since only a single datasource can be accessed by an entity, this display will show the members linked to the underlying MultiValue datasource:



DAC Datasource Specific Display

On clicking the Datasource specific radio button, the DAC maintenance screen changes its display content to show the definition details specific to the selected datasource.

This new display content is shown below:



The datasource specific display shows the base file and schema for the DAC. It also shows a checkbox named "Load all physical data". This check box allows you to indicate that even if a BAC tells the DAC that it is only interested in a subset of physical data, the entire physical data set will still be retrieved and passed back to the client. You should only uncheck this option if the underlying datasource record structure contains a lot of data. By only loading the physical data specified by the BAC, you will cut down on data volume size moving from datasource to client. You may also wish to uncheck this option if some of the data within the datasource record is highly sensitive in nature, in which case this option will allow you to prevent that data from ever being transmitted to the client.

The lower section of the datasource specific view is dedicated to allowing you to view and maintain the 5 aspects of the class definition, as described in the following sections.

Property Maintenance

The Properties tab provides 6 different views of the properties defined within the DAC.

The first view is a read-only summary display.

The middle 3 views provide the ability to define which property or properties are to be generated for each base datasource field. The base fields are broken down into nested data types, single (non-nested), multivalue nested and subvalue nested. An important concept to grasp at this point is the fact that when you have nested data, an entity, no matter which level of data nesting it represents, is able to contain properties that expose **any** data field within the record as a whole, single or nested. Therefore, you are always able to see the full list of Single-valued, Multi-valued and Sub-valued fields for the record as a whole.

The other views allow you to define information relating to specialized properties based on the presence of nested data as well as properties based around the need to amalgamate a property's value across a collection of instances.

The different property views are selected by clicking the relevant radio button at the top of the Properties tab area.

The 6 property views are described in the following sections.

Property View - Summary

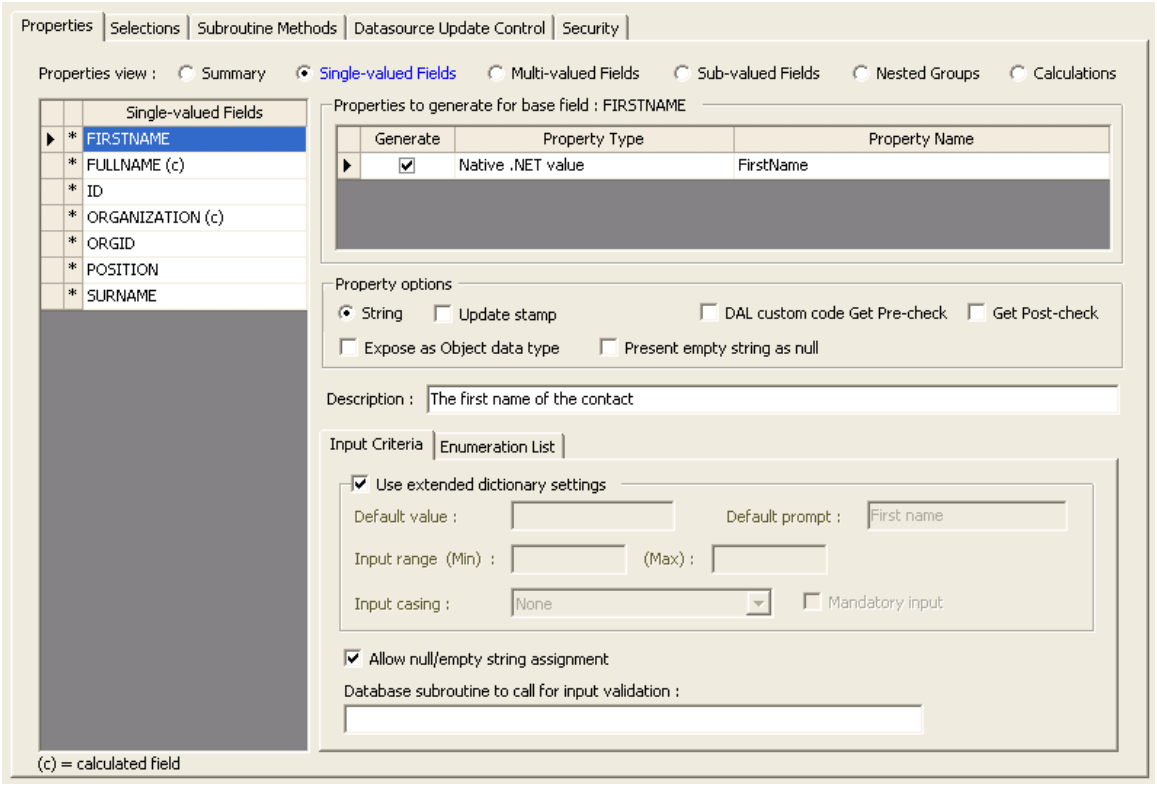
This view allows you to see the overall picture of which properties are currently defined for the class. If you double-click one of the rows within the displayed grid, you will be taken into one of the other views to see the details of that property definition.

Property View - Single-valued Fields

This view allows you to maintain the definition of which properties are to be included on the interface of the DAC based on the presence of single-valued fields within the base file.

Each single-valued base field within the base file can be represented by one or more properties.

The content of the Single-valued property view is shown below:



The grid on the left-hand side of the display allows you to select one of the single-valued base fields. Fields that represent calculated or dynamically derived non-physical data are suffixed with "(c)". Base fields that are currently represented by one or more properties are flagged with an asterisk in the second grid column.

The "Properties to generate" grid on the right-hand side of the display lists all of the available properties for the currently selected base field. The "Generate" column checkbox is used to indicate which of these available properties are to be actually included within the interface of the generated class code. The "Property Type" column displays the category of each potential property. The "Property Name" column allows you to enter the name of the property as it will appear within the generated code.

There are a number of possibilities for the Property Type column. The grid below describes each possible setting:

Property Type	Description
Native .NET value	A simple .NET data type. This will be one of the following standard .NET CLR data types: • String • Int32 • Int64 • Double • Decimal • DateTime • Boolean
String equivalent value	A string representation of either a date or time value.
MvArray of .NET values	An MvArray of .NET values. Please refer the Dynamic Array Representation of Data section below for more details on this topic.
MvArray of string equivalent values	An MvArray of string equivalent values, as described above. Please refer the Dynamic Array Representation of Data section below for more details on this topic.
Single related entity	A reference to a related singular entity instance.
Collection of related entities	A reference to a collection of related entity instances.
SmvArray of .NET values	An SmvArray of .NET values. Please refer the Dynamic Array Representation of Data section below for more details on this topic.
SmvArray of string equivalent values	An SmvArray of string equivalent values, as described above. Please refer the Dynamic Array Representation of Data section below for more details on this topic.

The area underneath the "Properties to generate" will change dynamically depending on the type of property currently selected. If a particular property has never been defined, the area will be empty; in which case clicking the Generate column checkbox will force the property definition input fields to appear.

Most property types will display a "Property options" group box beneath the properties grid. Again, the content of this area will change dynamically depending on the precise details of the currently selected property and its associated base field. Below is a grid indicating the range of possible input controls that may appear within the Property options groupbox area:

Input Field	Description
String	Indicates that the property value will be a String .NET data type. This input is automatically selected for base fields representing an alphanumeric value.
Int32/Int64	Indicates whether the property value will be either an Int32 or Int64 .NET data type. These inputs are displayed for base fields representing an integer numeric value.
Double/Decimal	Indicates whether the property value will be either a Double or Decimal .NET data type. These inputs are displayed for base fields representing a non-integer numeric value.
DateTime	Indicates that the property value will be a DateTime .NET data type. This input is automatically selected for base fields representing either a date or time value.
Boolean	Indicates that the property value will be a Boolean .NET data type. This input is automatically selected for base fields representing a Boolean value.
Update stamp	Allows you to indicate that the base field associated with this property holds a date-time modified update stamp. This input is only displayed for base fields representing an alphanumeric value. Only one property within the DAC as a whole may be selected as an update stamp. Update stamps are discussed in further detail in the Datasource Update Control Maintenance section below.
Exists check	Allows you to indicate that any value assigned to this property will be automatically validated to ensure that it is a valid existing primary key of an associated entity type. This option is only displayed for base fields flagged within the base file schema as holding a foreign key value.
Long format	This field is only displayed for "string equivalent value" property types and allows you to indicate whether the Windows "Long" date/time format should be used to format the returned date or time value. If this option is unchecked, the Windows "Short" format will be used.
Expose as Object data type	This checkbox is available for all Native .NET value property types (except Boolean). It allows you to expose the property as an Object data type as opposed to its .NET value type. Exposing a property has pros and cons. The advantage is that an Object data type gives you more flexibility in the way in which blank or bad datasource data is handled. This is because Solution Objects will adjust the

	actual type of data 'boxed' by the object to fit the retrieved data, including returning null values. The disadvantage is that you cannot guarantee the type of data returned by such a property.
Present empty string as null	This is available for any property which is exposed as an object data type as well as String data types. It results in an empty value (String.Empty) being presented as a null/Nothing value.
Related entity	Allows you select the type of entity that the reference held by a property represents. This field is only displayed for reference type properties.
Cascade deletes	Indicates that if an instance of the current entity is deleted, any instances of the entity type indicated by the "Related entity" input field that are held by a reference property will also be deleted. This field is only displayed for reference type properties.
Synchronize foreign keys	Indicates that if the entity type indicated by the "Related entity" contains a property holding the primary key of the current entity (i.e. a foreign key to the current entity), when the primary key of the current entity changes, any instances of the related entity held by a reference property will have this foreign key property updated. This field is only displayed for reference type properties.
DAL custom code Get Pre-check	Allows you to indicate that the PropertyGet function within the custom code module associated within the generated code module is to be called before the generated code retrieves the value of the property from the base data record. This allows you to control the value that is returned by the property and also to prevent the generated code from performing its default property value retrieval action. An example of where you may need to do this is where a base field's value is (either wholly or in-part) derived from the presence of a foreign key within the record. In this case, if the foreign key is blank, you may well want to avoid a pointless roundtrip to the datasource to derive the property value or return a custom value (from within your custom code) indicating a non-existing foreign key. Please refer to the Integrating Custom Code chapter for further details on this topic.

Get Post-check	Allows you to indicate that the PropertyGet function within the custom code module associated within the generated code module is to be called after the generated code retrieves the value of the property from the base data record. This allows you to control the value that is returned by the property if required. An example of where you may need to do this is where you wish to control the final formatting of a string property value. Please refer to the Integrating Custom Code chapter for further details on this topic.

Underneath the Property options groupbox area is an input field where you may enter a description of the field. This value will be initially set to the "Notes" section of the extended dictionary definition of the associated base field. The value of this field will be used in the XML-based IntelliSense help comments within the generated BAL code module.

At the foot of the property definition area is a tab control allowing you to enter further definition details.

The first tab, "Input Criteria", allows you to define the automatic property value formatting and validation that will be performed whenever a property value is retrieved or assigned. Input criteria is only available for Int32/64, Double/Decimal, DateTime and String property types representing physical data.

The fields presented in this tab are described in the following grid:

Input Field	Description
Use extended dictionary settings	Allows you to indicate that the settings to be used for formatting/validation are to be sourced at run-time from the extended dictionary definition of the associated base field. If this value if checked, the input fields that are covered by extended dictionary are disabled and display the settings currently held within the extended dictionary. Otherwise, the values must be set and maintained manually within this tab.
Default prompt	Used by some of the Solution Objects interface creation RAD tools as the default input label associated with an input control bound to this property.
Default value	When a new instance of an entity is created, this value will be set as the initial value of the property.
Mandatory input	Allows you to indicate that this property must contain a value before the entity data can be saved.
Input range	Allows you to indicate the minimum allowable value for this

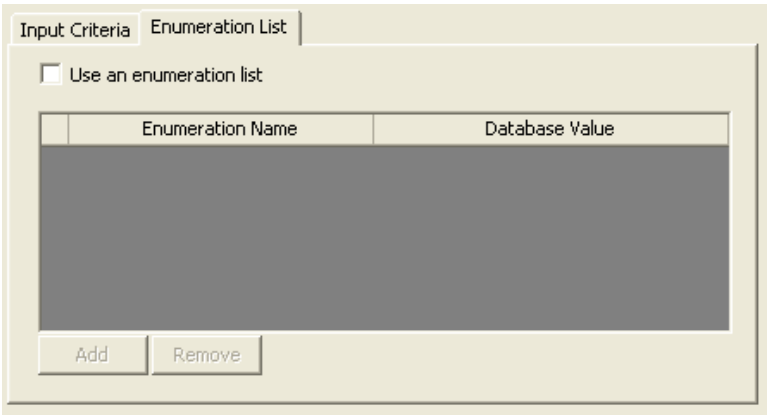
(Min)	property. This value is ignored for Boolean property types.
Input range (Max)	Allows you to indicate the maximum allowable value for this property. This value is ignored for Boolean property types.
Input casing	Allow you to specify the case conversion that will be automatically applied to any values assigned to this property. This value is ignored for Boolean property types.
Database subroutine	Allows you to specify the name of the database server resident subroutine to be automatically called in order to validate any value assigned to the property. This subroutine must contain 5 arguments in its signature as follows: **Arg#1** – The value for validation (this will be in external display format) **Arg#2** – The current item ID (may be blank for new instances) **Arg#3** – The (current) whole item string **Arg#4** – The value to be used (set on entry to the same value as Arg#1) – **must be in external display format – i.e. not in internal database format.** **Arg#5** – Validation error. Leave blank if validated OK An example of such a subroutine signature definition is as follows: `SUBROUTINE VALIDATE.PROPERTY (CANDIDATEVALUE, ITEMID, ITEMDATA, VALIDATEDVALUE, ERRORMSG)` Note, it is good efficient practice to clear the values of arguments 1, 2 and 3 on exit from the subroutine to optimize network traffic.
Use extended call signature	If this checkbox is ticked, the number of arguments that must be supplied to the specified database subroutine must be extended (as defined above) to include 3 extra arguments as follows: **Arg#5** – The name of the associated BAC and DAC. These 2 names are separated by a char 253 (VM). **Arg#6** – The name of the property being validated. **Arg#7** – The value of the property when the data was originally retrieved from the data source. **Arg#8** – Validation error. Leave blank if validated OK An example of such a subroutine signature definition is as follows: `SUBROUTINE VALIDATE.PROPERTY (CANDIDATEVALUE, ITEMID, ITEMDATA, VALIDATEDVALUE, ENTITYNAMES, CURRENTVALUE, PROPERTYNAME, ERRORMSG)`

	Note, it is good efficient practice to clear the values of arguments 1, 2, 3, 5, 6 and 7 on exit from the subroutine to optimize network traffic.

The second tab, "Enumeration List", allows you to define whether the list of potential values for this property are to be represented as an enumeration type. This will ensure that only a finite set of values can ever be assigned to the property.

Enumeration lists are only available for Int32/64, Double/Decimal and String property types with the exception that they are not available for properties representing foreign keys. If the enumeration list option is not available, the Enumeration List tab content will be blank.

If allowed, the enumeration tab displays the following content:



The top checkbox allows you to indicate that an enumeration list is to be used.

The grid and buttons beneath allow you to add/edit/remove enumeration entries. Each enumeration entry comprises 2 values:

Enumeration name - The name of the enumeration item as it will appear in the generated code.

Database value - The raw database field value that this enumeration item represents.

Note, an additional enumeration item named "UnknownValue" is automatically generated as part of the enumeration type. This value will be chosen if the database record contains a value which is not contained within the specified list of enumeration database values.

Property View - Multi-valued Fields

This view allows you to maintain the definition of which properties are to be included on the interface of the class based on the presence of multivalued fields within the base file.

The content and operation of this view is very similar to that of the Single-valued view with the exception that a number of additional property types may appear in the Properties to generate grid and some additional input fields may be present within the Property options groupbox area.

The potential additional Property Types are listed in the following grid:

Property Type	Description
String-based text version	A string representation of a multivalued field value. Internal multivalue separators will be replaced with a custom defined character/string.
SvArray of .NET values	An SvArray of .NET values. Please refer the Dynamic Array Representation of Data section below for more details on this topic.
SvArray of string equivalent values	An SvArray of string equivalent values. Please refer the Dynamic Array Representation of Data section below for more details on this topic.

The additional Property option input fields are listed in the following grid:

Input Label	Description
VM	The character(s) to be used to replace value marks within a text representation property type. If left blank, CRLF will be used.
SVM	The character(s) to be used to replace subvalue marks within a text representation property type. If left blank, ";" (semi-colon) will be used.

Property View - Sub-valued Fields

This view allows you to maintain the definition of which properties are to be included on the interface of the class based on the presence of multivalued fields within the base file.

The content and operation of this view is very similar to that of the Multi-valued view with the exception that a number of additional property types may appear in the Properties to generate grid.

The potential additional Property Types are listed in the following grid:

Property Type	Description
SmvArray of .NET values	An SmvArray of .NET values. Please refer the Dynamic Array Representation of Data section below for more details on this topic.
SmvArray of string equivalent values	An SmvArray of string equivalent values. Please refer the Dynamic Array Representation of Data section below for more details on this topic.

Property View - Nested Groups

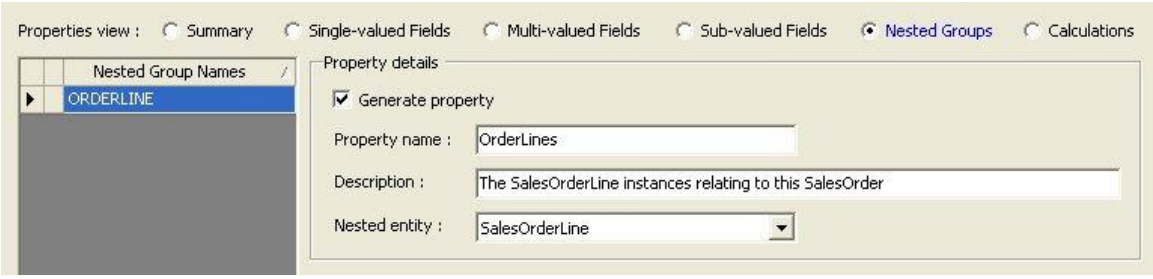
This view allows you to define properties that represent the presence of nested data within a base file's record structure, for example, an OrderLines property representing a MultiValue group of order line details. Nested group properties are only available for non-nested and multivalue nested entities.

Each nested group property will contain a reference to the entity(s) representing the nested data at the next nested-level down from the entity's own nested data level. Thus, for non-nested (top-level) entities this means that each multivalue group within the record structure may be represented by a nested group property; this is dependent, of course, on there actually being an entity defined within the model which is mapped to the mv group in question.

For entities representing multivalue nested data, each subvalue group within the associated multivalue group may be represented by a nested group property.

Nested group properties, by definition, always manifest themselves as collective instances of the related entity.

The content of the Nested Groups property view is shown below:



For non-nested data (top-level) entities the Nested Group Names grid displays the multivalue groups present within the record structure. For entities representing multivalue nested data the Nested Group Names grid displays the subvalue groups present within the multivalue group associated with the entity.

The Property details groupbox area to the right of the grid allows you to enter various pieces of definition information as described in the grid below:

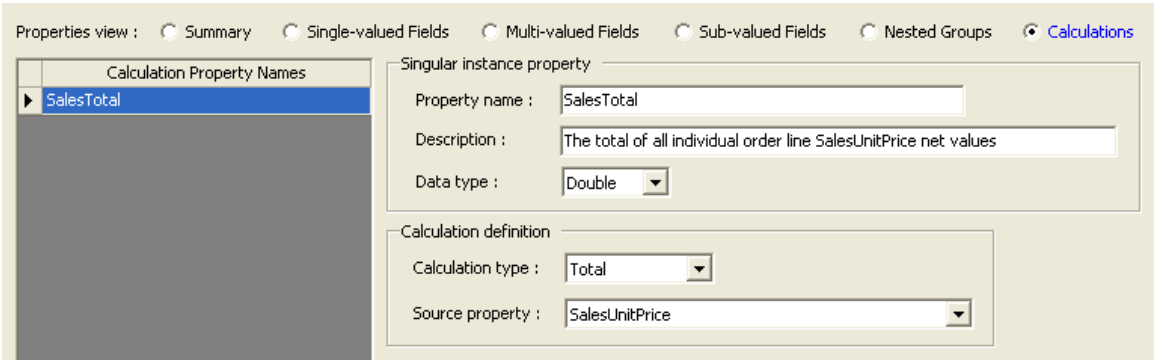
Input Field	Description
Generate property	Indicates whether a property is to be generated for the currently selected group.
Property name	Name of the property as it will appear within the generated code.
Description	Description of the property. The value of this field will be used in the XML-based IntelliSense help comments within the generated BAL code module.
Nested entity	The entity that is mapped to the currently selected group.

Property View - Calculations

This view allows you to define properties that represent a calculation based upon all values of a given property within a collective instance of the entity.

A calculated property will only return a value if the entity instance being referenced belongs to a collective instance. If this is the case, each instance within the collection returns exactly the same value for each calculation property (except for a percentage calculation, which will, of course, return a potentially different percentage value for each instance). This feature is particularly useful in data binding where an input control may be bound to a calculation property and used to display, for example, a running total of a particular property within the collection of entities. As the value of the property in any of the collection instances changes, the value of the calculated property changes and will be automatically updated and redisplayed.

The content of the Calculations property view is shown below:



The grid on the left-side of the display lists all of the calculation properties currently defined. The input areas to the right of the grid allow you to maintain

the definition of the currently selected calculation property. The Add and Remove button can be used to add and remove entries from the grid.

The Singular instance property groupbox area allows you to enter various pieces of definition information as described in the following grid:

Input Field	Description
Property name	Name of the property as it will appear within the generated code.
Description	Description of the property. The value of this field will be used in the XML-based IntelliSense help comments within the generated BAL code module.
Data type	The .NET data type of the property.

The Calculation definition groupbox area allows you to enter additional definition information describing the nature of the required calculation as described in the grid below:

Input Field	Description
Calculation type	The type of calculation to be performed.
Source property	The property to supply the value to be used in the calculation. Only numeric-based properties are listed here.

Dynamic Array Representation of Data

Because Solution Objects is primarily a MultiValue database oriented tool, it is logical and necessary to be able to present multivalued data in its native form, i.e. the traditional MultiValue dynamic array. For this reason, Solution Objects, by means of one of the available property types, allows you to expose multivalue data as a dynamic array.

There are 4 implementations of dynamic arrays provided by Solution Objects, as described in the grid below:

Array Type (Name)	Description
MvArray	Holds multivalued data.
SvArray	Holds subvalued data.
SmvArray	Holds submultivalued data (2 dimensional)

SmvDynamicArray	Holds submultivalued data (2 dimensional). This implementation is for use by developers who wish to use a dynamic array in a standalone manner, as opposed to the first 3 which are used as property types within the context of DAC/BAC code generated by Solution Objects.

Each array implementation comes in the form of a Generic class, i.e. each array is declared in a strongly-typed manner in order to provide better performance and reduce run-time errors relating to data type mismatches.

The interface of each implementation is very similar and is fully documented in the form of XML-based IntelliSense help within Visual Studio.

As an example, if there is a field which holds the lines of an address as a series of multivalues and this field is exposed as an MvArray property called "AddressLines", the following code could be used to access the 2nd line of the address:

VB:

```
Dim addressLine2 As String = MyEntity.AddressLines(2)
```

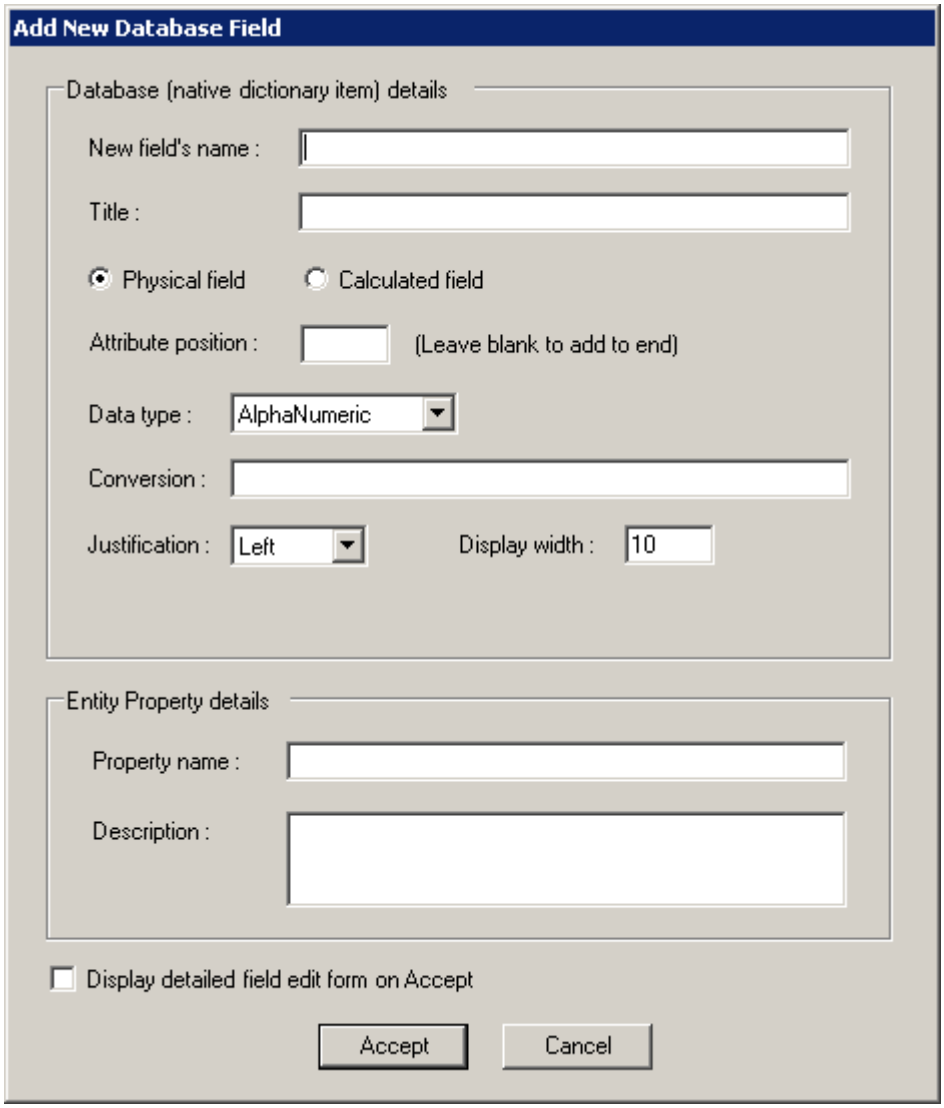
C#:

```
string addressLine2 = MyEntity.AddressLines[2];
```

Maintaining Datasource Schema

The Data Access Class maintenance window allows you to maintain the underlying schema upon which the entity is based. To do this right-click the relevant row within the Fields grid (on any of the Single, Multi-valued or Sub-valued views) and select the required maintenance option (edit, add or delete).

Adding a new schema field requires you to supply a number of datasource specific details:



The upper half of this window allows you to enter the details of the native dictionary item of the field. The lower half allows you to enter the name of the first property (others may be possible depending on the field type) along with a description of the property.

If the "Add New Field" option is selected for a the Multi-valued or Sub-valued Fields grid, you are able to specify the multivalue and (where relevant) the subvalue group that the new field is to belong to. Existing group names are displayed via a ComboBox control or you can type in a new group name to create a new group.

Selecting the "Edit Selected Field" option from the right-click menu of the Field grid displays a window which allows you to edit both the native and extended dictionary details associated with the field. For more information on this please

refer to the "Extended Dictionary Definitions" chapter of the Core Objects Developer Guide.

Selection Method Maintenance

Solution Objects supports the concept of "Selection Methods". Each selection method is represented as a function method on the interface of the DAC and subsequently on one of more of the BACs that derive from it. Some selection methods may optionally be exposed as properties instead of methods.

Selection methods allow you to predefine and abstract a variety of selection actions relating to the entity. Users (developers) of the resulting BAL will be able to utilize these selection methods to invoke the selection actions that you define without any knowledge of the underlying datasource or mechanism by which the selection is being executed.

In addition to being able to define custom selection actions, you are able to include 2 standard selection methods as static (Shared) methods on the DAC. The 2 checkboxes at the top of the Selections tab allow you to indicate whether you wish to include either of these 2 standard selection methods, which are described in the grid below:

Selection Method Name	Description
SelectAll	Allows the entire contents of the base file to be selected.
SelectGeneric	Allows the contents of the base file to be selected/sorted based on datasource specific select/sort command syntax.

The Selections tab provides 3 different views of the selection methods defined within the DAC.

The first view is a read-only summary display.

The last 2 views provide the ability to define 2 different types of selection methods.

The 3 selection method views are described in the following sections.

Selections View - Summary

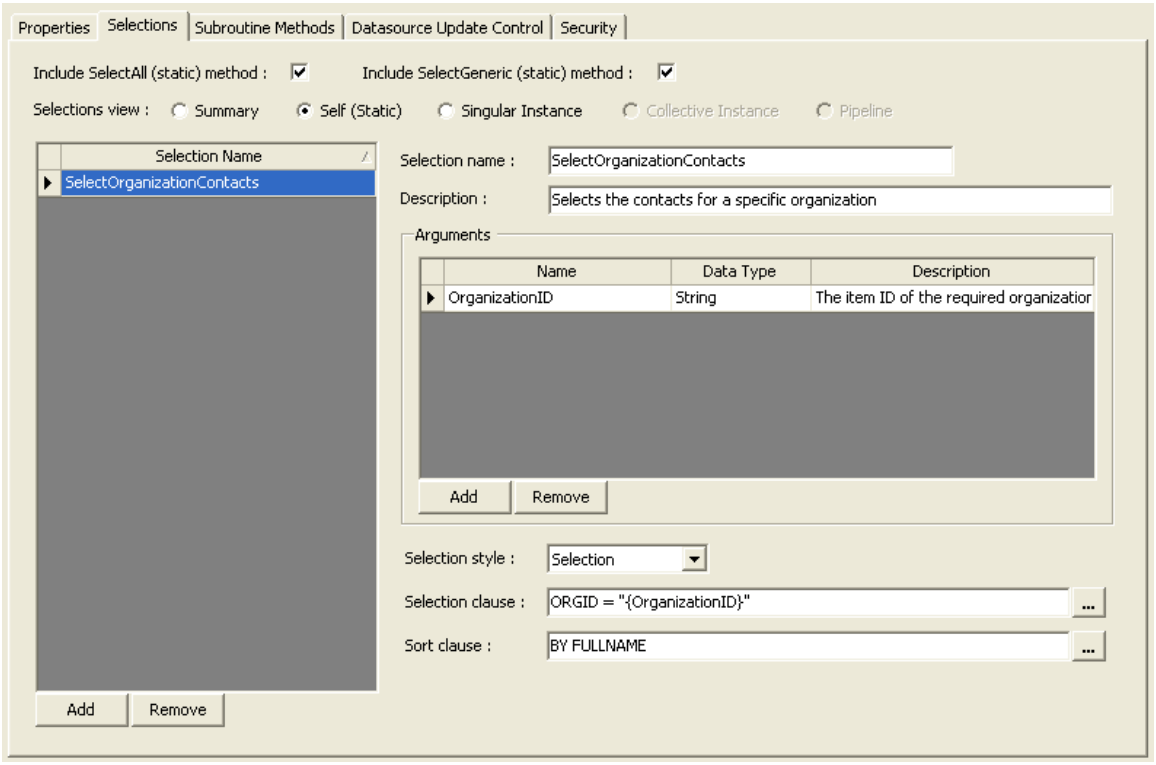
This view allows you to see the overall picture of which selection methods are currently defined for the class. If you double-click one of the rows within the

displayed grid, you will be taken into one of the other views to see the details of that selection method definition.

Selections View – Self (Static)

This view allows you to maintain the selection methods that select instances of the entity itself. That is, the return value of a Self selection method is a collective instance of the entity on whose interface this selection method is defined.

The Self selection view shows the following display:



The grid on the left of the display lists the current Self selection methods. The region to the right of the grid allows you to maintain the currently selected method.

Each Self selection method can be defined to require a number of arguments to be passed as part of its call signature. The Arguments groupbox area allows you to specify each of these arguments as required.

The grid below describes the input fields used to define a Self selection method:

Input Field	Description
Selection name	Name of the selection method as it will appear within the generated code.
Description	Description of the selection method. The value of this field

	will be used in the XML-based IntelliSense help comments within the generated BAL code module.
Arguments - Name	The name of a required argument value as it will appear within the generated code.
Arguments - Data Type	The .NET data type of the argument parameter.
Arguments - Description	Description of the purpose or content of the argument. The value of this field will be used in the XML-based IntelliSense help comments within the generated BAL code module.
Selection style	Allows you to choose the general style of the selection method: • **Selection** - A supplied select command will be used to select items from the datasource. • **Subroutine** - A database resident subroutine will be used to perform the selection. • **SingleItem** - Only a single item will be selected. • **SingleMultivalue** - Only the contents of a single multivalue position will be selected • **SingleSubvalue**- Only the contents of a single subvalue position will be selected Each of these different selection styles (and their constituent definition input fields) are described in the following sections below.

Select Selection Style - Selection

This selection style allows you to enter a selection and sort clause using the native syntax of the underlying datasource. The names of any arguments may be inserted into these clauses in order to insert run-time supplied values as required.

In the above screenshot, the supplied value of the {OrganizationID} argument will be inserted into the select clause at run-time, thus allowing the contacts for one specific organization to be selected.

Select Selection Style - Subroutine

This selection style allows you to specify a database resident subroutine to be used to perform the selection. Note, this option is only available for entities that represent non-nested data.

The subroutine that you specify here must contain 7 arguments in its call signature as follows:

Arg#1 – *(input arg)* The name of the selection method.

Arg#2 – *(input arg)* The calling context. If you wish to pass in a static piece of data (string) specific to a particular selection method into a subroutine this string may be specified after the subroutine name within the DAC definition. For example, if the Subroutine name field is set as "GETSAVELIST CONTACT123", then GETSAVELIST is the name of the subroutine and its 2nd argument will be set to "CONTACT123" on being called.

Arg#3 – *(input arg)* An attribute mark separated list of argument names. This corresponds to the argument names defined for the selection method.

Arg#4 – *(input arg)* An attribute mark separated list of argument values. This corresponds to the arguments defined for the selection method.

Arg#5 – *(input arg)* The sort specification (if supplied).

Arg#6 – *(output arg)* Indicates the return style of the selected list of item IDs. This argument must be set to one of the following values:

-1 = No items selected. Additional descriptive information can be optionally provided in Arg#7

0 = An attribute mark separated list of item IDs contained in Arg#7

1 = An active select variable contained in Arg#7

2 = A saved list, the name of which is held in Arg#7

Any other value will be regarded as an error message and the selection action will be cancelled with an exception being thrown to the end application.

Arg#7 – *(output arg)* The selected item IDs, the format of which will be indicated by Arg#5.

An example of a subroutine call signature definition would therefore be:

```
SUBROUTINE CONTACT.SELECT.CHIEFS (SELECTIONNAME, CONTEXT, SUBARGNAMES, SUBARGVALUES,
SORTSPEC, RETURNSTYLE, ITEMIDS)
```

Select Selection Style - SingleItem

This selection style is useful if all of the required entity instances can be retrieved with a single item read of the base file. This will be much more efficient than performing a selection specifying the required single item ID. It can be useful if any of the following types of selections are required:

- A selection list consisting of only a single non-nested entity instance. For example, in a data bound form where the item ID is either entered by the user or passed in from another section of code.
- A selection list consisting of only the multivalue nested entities relating to a single parent entity, for example, in the SOP demo database all the order lines relating to just a single sales order.
- A selection list consisting of only the subvalue nested entities relating to a single top-level parent entity, for example, in the SOP demo database all the deliveries relating to just a single sales order.

This selection style requires that you specify only a single argument which will be assumed to contain the required item ID.

Select Selection Style - SingleMultivalue

This selection style is only relevant for entities that represent either multivalue or subvalue nested data and is useful if all of the required entity instances can be retrieved with a single item read of the base file. This will be much more efficient than performing a selection specifying the required single item ID. It can be useful if any of the following types of selections are required:

- A selection list consisting of only a single multivalue nested entity, for example, in the SOP demo database a specific order line.
- A selection list consisting of only the subvalue nested entities relating to a single multivalue within a single top-level parent entity, for example, in the SOP demo database all the deliveries relating to just a order line.

This selection style requires that you specify 2 arguments, the first of which will be assumed to contain the required top-level entity item ID, the second of which will be assumed to contain the required multivalue position.

Select Selection Style - SingleSubvalue

This selection style is only relevant for entities that represent subvalue nested data and is useful if all of the required entity instances can be retrieved with a single item read of the base file. This will be much more efficient than performing a selection specifying the required single item ID. It can be useful if a selection list consisting of only the subvalue nested entity relating to a single subvalue within a single multivalue within a single top-level parent entity is required, for example, in the SOP demo database a single order delivery.

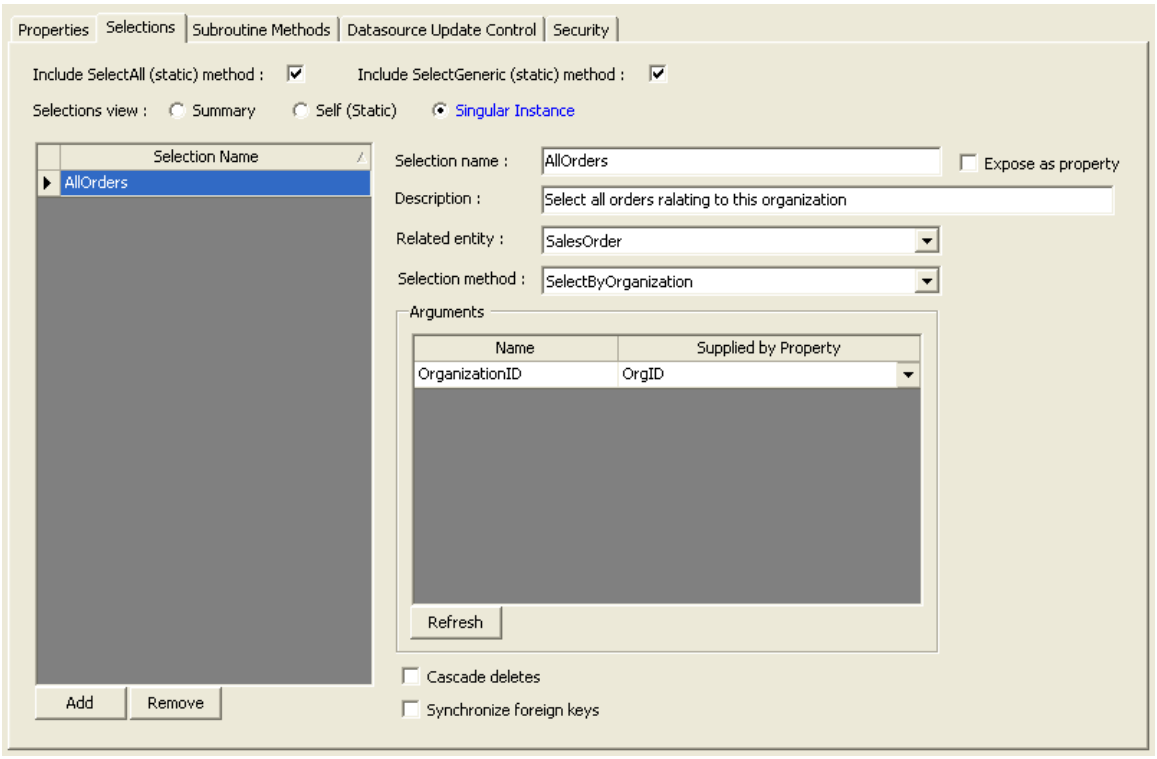
This selection style requires that you specify 3 arguments, the first of which will be assumed to contain the required top-level entity item ID, the second will be

assumed to contain the required multivalue position and the 3rd argument will be assumed to contain the required subvalue position.

Selections View – Singular Instance

This view allows you to maintain the definition of selection methods that select one or more entity instances that in some way relate to a single instance of the same or a different entity type. Because this type of selection method is an instance method (as opposed to Self selection methods which are static methods on the class) the selection process can, if required, use property values from the source entity instance.

The Singular Instance selection view shows the following display:



As in Self selection methods, the grid on the left of the display lists the current Singular Instance selection methods. The region to the right of the grid allows you to maintain the currently selected selection method.

The Selection name and Description input fields are as per Self selection methods.

The Expose as property checkbox allows you to control the way in which the method is manifested on the interface of the class. This subject is discussed in detail in the following section.

All Singular Instance selection methods need to identify the type of entity to be selected along with a Self selection method defined for that entity type. Based on

the Self selection method that is selected, the list of arguments defined for that method are listed in the Arguments grid. You can then select for each of these arguments the property from this entity instance that is to supply the argument value at run-time. If no property name is specified for an argument, the argument value will appear on the call signature of the Singular Instance selection method.

Singular Instance selection methods are often used to represent parent-child relationships and so, for this selection method type, you are able to indicate (by ticking the Cascade deletes checkbox) that when a parent instance is deleted, the child entities (as defined by the result of this selection method) are also to be deleted.

The Synchronize foreign keys option is only relevant if the child entities contain a foreign key property pointing back to the parent entity. If this is the case and if this option is ticked then if the primary key of the parent entity changes (as will be the case, for example, when the primary key of a new parent entity instance is assigned via auto ID generation on update) the foreign key property of all child entities will be set to the new primary key of the parent.

Exposing Selection Methods as Properties

A single instance selection method may be represented as either a method or a property on the interface of the class.

The main advantage of exposing it as a property is that you can use it directly with Visual Studio data binding tools due to the fact that is will be included in the list of bindable properties within the Data Sources window.

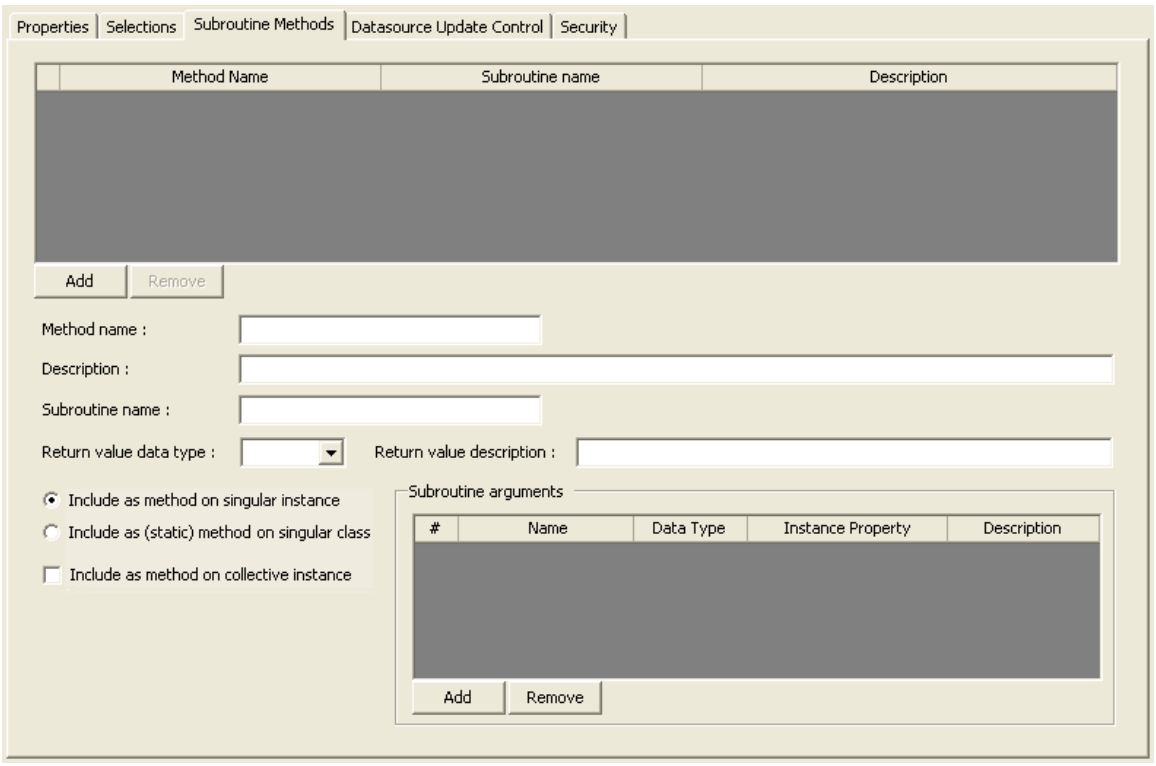
Conversely, exposing it as a method provides you (by the presence of multiple overloads to the method) with slightly more control as how the selection method is executed.

Please refer to the chapter on Reading and Selecting Data for more details on the control offered by the various overloads of a selection method.

Subroutine Method Maintenance

The 3rd possible type of member on the interface of a DAC is a method that is associated directly with a database resident subroutine. These are called Subroutine Methods.

The Subroutine Methods tab allows you to maintain the definitions of these types of methods as shown below:



In the above display, the top grid displays the currently defined subroutine methods. The bottom area of the tab area allows you to maintain the definition of the currently selected subroutine method as described below:

Input Field	Description
Method name	Name of the subroutine method as it will appear within the generated code.
Description	Description of the subroutine method. The value of this field will be used in the XML-based IntelliSense help comments within the generated BAL code module.
Subroutine name	Name of the database resident subroutine to be called.
Return value data type	.NET data type of this method's return value. This may be set to "(None)" if the method is to be declared as having no return value.
Return value description	Description of the subroutine's return value. The value of this field will be used in the XML-based IntelliSense help comments within the generated BAL code module.
Include as method on singular instance	Indicates if the subroutine method is to be declared as an instance method on the singular class.

Include as (static) method on singular class	Indicates if the subroutine method is to be declared as a static (class-level) method on the singular class.
Include as method on collective instances	Indicates if the subroutine method is to be declared as a static (class-level) method on the collective class.

The Subroutine arguments grid allows you to define each argument that the database subroutine requires. The arguments in this grid must appear in the same order as that declared on the call signature of the subroutine. The cells of each argument row within the grid are described below:

Column Heading	Description
Name	The name of a required argument value as it will appear within the generated code.
Data Type	The .NET data type of the argument parameter.
Instance Property	Indicates the property that is to provide the value at run-time for this subroutine argument. If this cell is left blank, the argument will be included on the call signature of the subroutine method.
Description	Description of the purpose or content of the argument. The value of this field will be used in the XML-based IntelliSense help comments within the generated BAL code module.

If the subroutine method is defined as having a return value then an extra argument will be automatically added behind the scenes at the end of the call signature by Solution Objects when calling the subroutine. Thus, for example, if you define a subroutine method with 2 arguments and a return value, the actual database subroutine must be defined with 3 arguments – the final argument being treated by Solution Objects as the return value of the subroutine.

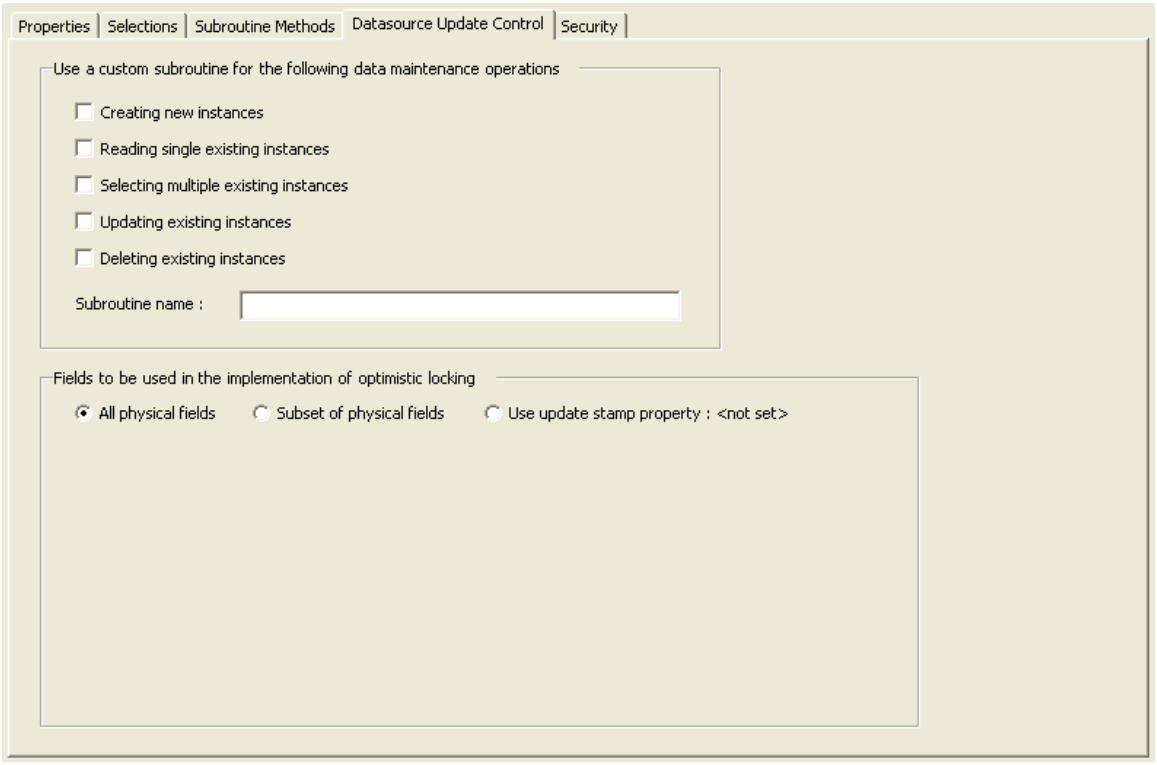
Datasource Update Control Maintenance

The Datasource Update Control tab allows you to maintain 2 aspects relating to the control of how Solution Objects interacts with the underlying datasource for this DAC:

1. The use of a custom database resident subroutine to handle one or more types of database interaction.

2. The specification of which properties are significant in the context of record updating when optimistic locking is being used.

The content of the Datasource Update Control tab is shown below:



Note, the "Reading single existing instances" option also controls existence checking actions.

Also note, custom datasource updating is not available for entities that represent nested data.

Custom Datasource Update Control

The top section allows you to indicate which database actions are to be handled by your custom subroutine as identified in the Subroutine name input field. This subroutine will be called in place of the standard Solution Objects code base.

Please note however that Subroutine-based Selection Methods and Subroutine Methods will still be processed as normal (i.e. using the defined subroutine) and will not use the custom datasource subroutine

The subroutine that you specify here must contain 8 arguments in its call signature. The usage of these 8 arguments varies slightly depending on the type of database action being performed, however the use of the first 2 and last argument is constant:

Arg#1 - *(input arg)* The singular name of the associated DAC and BAC. These 2 names are separated by a char 253. For database action type 'S' (see below) this argument also holds the name of the selection method being used (separated from the singular names by a char 254).

Arg#2 - *(input arg)* Database action type indicator. This argument will be set to one of the following values:

C = Create a new database item.

R = Read a single existing database item.

E = Check if a given database item currently exists.

S = Select a list of existing database items based on a selection method.

I = Select a list of existing database items based on a list of item IDs. (This selection style is used internally by Solution objects).

U = Update (write) a database item.

D = Delete a database item.

O = On-demand data field value recalculation.

Arg#8 - *(output arg)* Error message text. Empty string indicates no error.

The grids below document the usage of arguments 3 through 7 for the various database action types. Arguments 3 through 6 are input arguments. i.e. their values will be set on entry by Solution Objects. Argument 7 (for those types where it is relevant) is an output argument and its value should be set by your subroutine.

Database action type : C - Create a new database item

Argument #	Description
3	The relevant item ID. This argument will only contain a value on entry if a primary key value has been supplied by the calling client code and needs to be set on exit to the ID of the new item (if relevant).
4	Lock type indicator. Will be one of the following values: "N" : No lock required "P" : Pessimistic lock required "O" : Optimistic lock required Note, if a pessimistic lock cannot be obtained, Arg#8 should be set to an appropriate error message.
5	n/a - passed as blank string.
6	n/a - passed as blank string.
7 (Output)	Attribute mark (char 254) separated list of item data elements. Note, the order of data elements within this argument needs to be as per indicated by the base fields used within the DAC definition.

Database action type : R - Read a single existing database item

Argument #	Description
3	The relevant item ID.
4	Lock type indicator. Will be one of the following values: "N" : No lock required "P" : Pessimistic lock required "O" : Optimistic lock required Note, if a pessimistic lock cannot be obtained, Arg#8 should be set to an appropriate error message.
5	n/a - passed as blank string.
6	n/a - passed as blank string.

7 (Output)	Attribute mark (char 254) separated list of item data elements. Note, the order of data elements within this argument needs to be as per indicated by the base fields used within the DAC definition.

Database action type : E - Check if a given database item currently exists

Argument #	Description
3	The relevant item ID. This argument may contain more than one item ID. If this is the case each will be separated by a value mark (char 253) character.
4	n/a - passed as blank string.
5	n/a - passed as blank string.
6	n/a - passed as blank string.
7 (Output)	'0' to indicate that the item does not exist or '1' to indicate that it already exists. If more than one item ID has been supplied in Arg#3, this should be a value mark (char 253) separated list of 1's and 0's corresponding with the supplied list.

Database action type : S - Select database items based on selection criteria

Argument #	Description
3	The selection clause of the underlying static (self) selection method.
4	The sort clause of the underlying static (self) selection method.
5	An attribute mark (char 254) separated list of argument names. Corresponds to the argument names defined for the selection method being invoked.
6	An attribute mark (char 254) separated list of supplied argument values. Corresponds to the argument names defined for the selection method being invoked.
7 (Output)	List of selected items. This needs to be a series of item data blocks as per Type R, with each block separated by char(30) and with the item ID of each item added to the front the item followed by an attribute mark.

Database action type : I - Select database items based on item IDs list

Argument #	Description
3	n/a - passed as blank string.
4	n/a - passed as blank string.
5	Attribute mark (char 254) separated list of required item IDs.
6	n/a - passed as blank string.
7 (Output)	List of selected items. This needs to be a series of item data blocks as per Type R, with each block separated by char(30).

Database action type : U - Update (write) a database item

Argument #	Description
3	The relevant item ID. If the item ID is adjusted by the routine this argument needs to be set on exit to the new ID.
4	The record contents (attribute mark separated) as currently held in memory on the client.
5	n/a - passed as blank string.
6	n/a - passed as blank string.
7	n/a - passed as blank string.

Database action type : D - Delete a database item

Argument #	Description
3	The relevant item ID.
4	The record contents (attribute mark separated) as currently held in memory on the client. Note, this argument will only contain a value if the instance-based overload of the Delete method has been invoked.
5	n/a - passed as blank string.
6	n/a - passed as blank string.
7	n/a - passed as blank string.

Database action type : O - On-demand data field value recalculation

Argument #	Description
3	The relevant item ID.
4	The record contents (attribute mark separated) as currently held in memory on the client.
5	Attribute mark (char 254) separated list of dictionary item names requiring value recalculation.
6	n/a - passed as blank string.
7 (Output)	Attribute mark (char 254) separated list of field data values. Note, the order of data elements within this argument needs match the list of dictionary names as passed in Arg#5

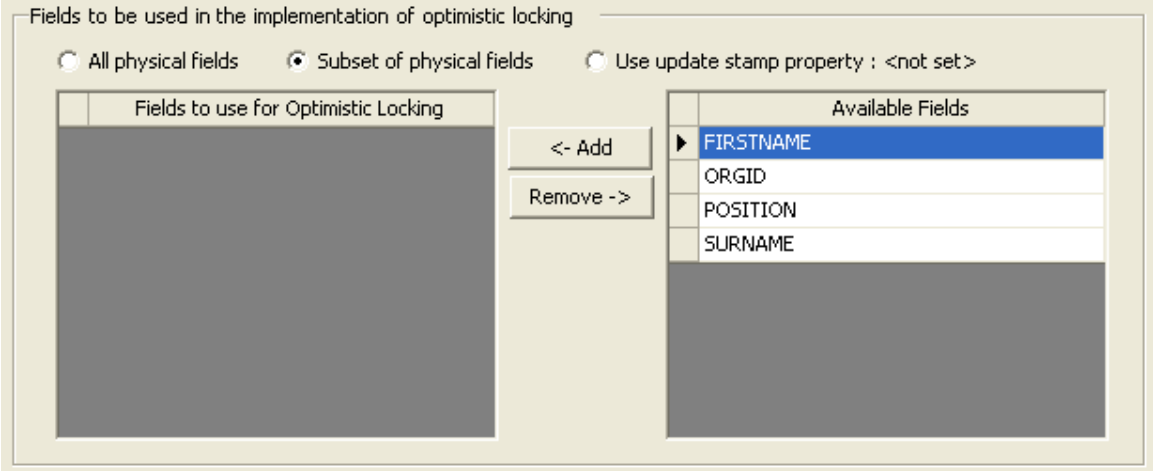
Optimistic Locking Control

If optimistic locking is specified as part of a read or selection action, Solution Objects needs to know which fields within the record are to be considered as "significant" when it comes to comparing the initial read image of a record with the current database image at the time of update commit.

The lower half of the Datasource Update Control tab allows you to define which fields are "significant" in this context.

The default setting is "All physical fields". This results in the entire data content of the record being used to determine whether an update commit is allowed to proceed.

The "Subset of physical fields" option allows you to restrict the comparison process to use only a subset of physical fields. When this option is selected, the following display is shown:



This allows you to define the required (physical) field subset.

The third optimistic locking option is "Use update stamp property". Each non-primary/foreign key physical property with a data type setting of "String" may be flagged as holding an update stamp. Only one such property within the DAC as a whole may be chosen to perform this role. If a property is currently identified as an update stamp property its name will be shown alongside the Use update stamp property radio button, otherwise "<not set>" will be displayed - as illustrated above.

The content of the update stamp property will be controlled by Solution Objects, and comprises a string with the following structure:

{timestamp}_{guid}_(username}

Where:

{timestamp} is a client-based system time stamp of the format:
yy.MM.dd.HH.mm.ss.ii (ii = milliseconds)

{guid} is an eight character GUID

{username} is the client-based (Windows) user name

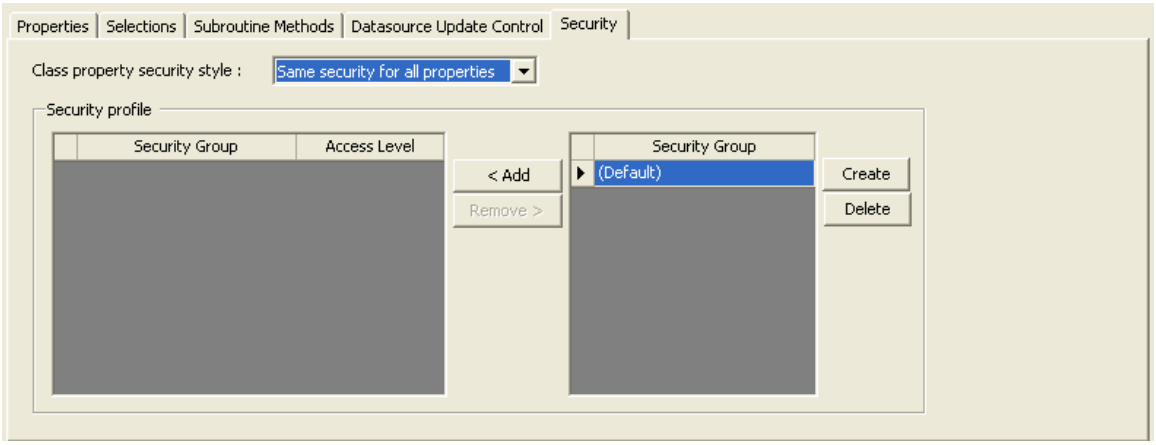
Security Maintenance

The Security tab allows you to maintain the security group profile of one or more properties within the DAC.

The security mechanism to which this information relates is described in detail within the User-based Property Security chapter.

The initial setting of the Class property security style will default to "No security" unless it has been set otherwise by the entity generation process.

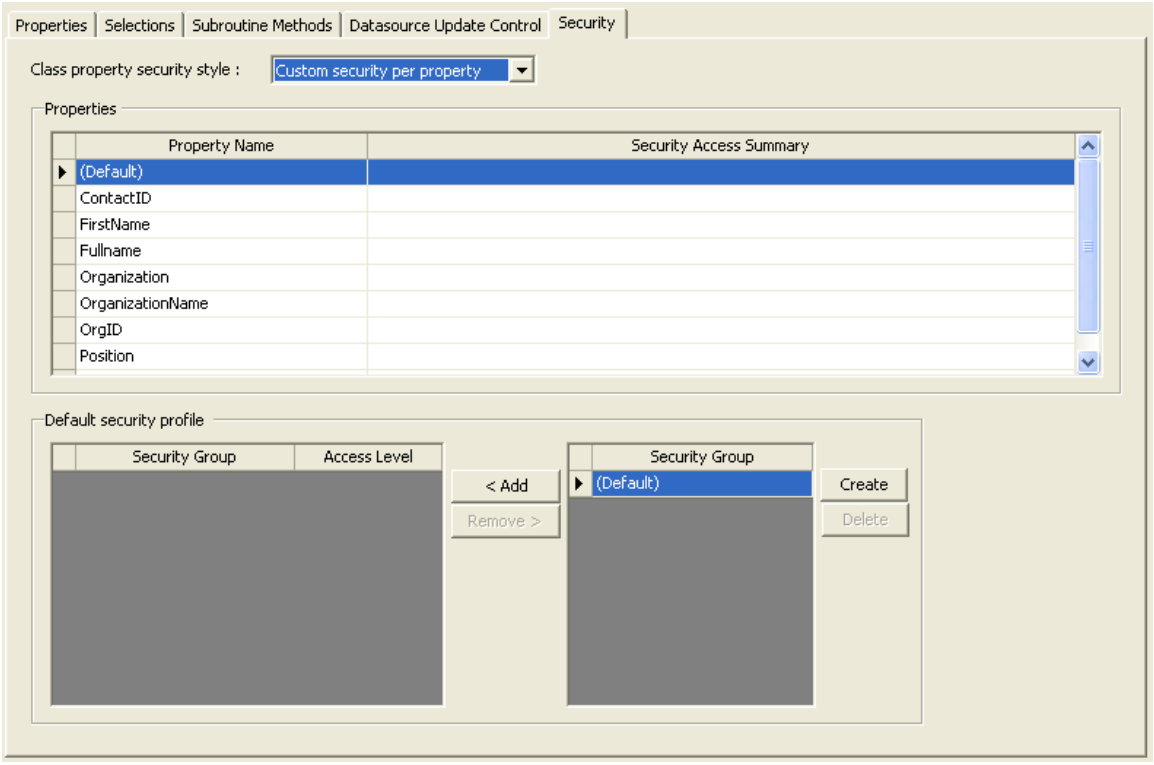
If the "Same security for all properties" option is selected then all properties within the DAC will be given a security profile as defined by the following display:



This allows you to add items from the Security Group grid on the right into the profile grid on the left. You can also add new or remove existing security groups by using the Create and Delete buttons (note, the list of security groups is held at entity model version level).

Entries added to the profile grid on the left may have their Access Level set as required: None, Read-only or Full.

If the "Custom security per property" option is selected then each property within the DAC may be given a unique security profile as defined by the following display:



The top half of this display allows you to select a property. The "(Default)" property name is used to define the security profile for all properties that do not have a security profile set explicitly.

The bottom half of the display allows you to specify the security profile for the currently selected property and works in exactly the same manner as per the "Same security for all properties" option described above.

Schema Mismatch Warnings

On entering the DAC maintenance screen, Solution Objects will automatically compare the schema details currently held within the DAC definition with those currently defined for its base file. If any mismatches are detected a warning message will be given.

Some mismatches can be resolved automatically by Solution Objects – if any of these types are found a message is displayed that identifies the mismatches and informs the user that the automatic adjustment(s) to the DAC definition will be applied on next save of the DAC.

If any mismatches are found that cannot be resolved automatically you will need to exit from the DAC maintenance screen and to run the "Validate Data Access Class Definition" menu option (from the Data Access Class node's right-click context menu) and a message to this effect is displayed by the DAC maintenance screen.

Please refer to the Validating Entity Definitions chapter for full details on validating DAC definitions.

Maintaining Business Access Class Definitions

This chapter describes how you are able to maintain the definition of Business Access Classes associated with an Entity .

Overview

Each entity may have one or more Business Access Classes (BAC) defined. A BAC is always based on the DAC definition for that entity.

Ultimately, the code generation process of Solution Objects will allow you to group together (as required) a collection of BACs and present them collectively within a Business Access Layer (BAL) assembly. It is the BAL assembly (and consequently the constituent BACs within that assembly) that application developers reference and utilize within their application.

The purpose of a BAC, therefore, is to refine or restrict the interface or functioning of the underlying a DAC such that a class that is more suited or efficient for the purposes of a given community of developers can be produced.

Reasons supporting the concept of a BAC include:

1. It allows a complicated or very rich DAC interface to be presented in a simplified manner.
2. It allows properties that represent sensitive or confidential data to be omitted from the interface of the class that is ultimately presented to application developers, thus denying them access to that data.
3. It allows properties that represent very large quantities of data to be omitted from the interface of the class that is ultimately presented to application developers, thus preventing them from accidently triggering very large data movements from datasource to client and vice-versa.

Each BAC identifies the following:

1. The subset of DAC members (properties, selection methods and subroutine methods) that are to be included on its interface.
2. The properties that are to be made available in a read-only manner.
3. The sections of data within the DAC that are to be retrieved as part of the initial data retrieval process.

Creating a New BAC

Each entity node within the Data Manager's treeview has a sub-node named "Business Access Classes". It is within this node that the BACs for an entity are listed.

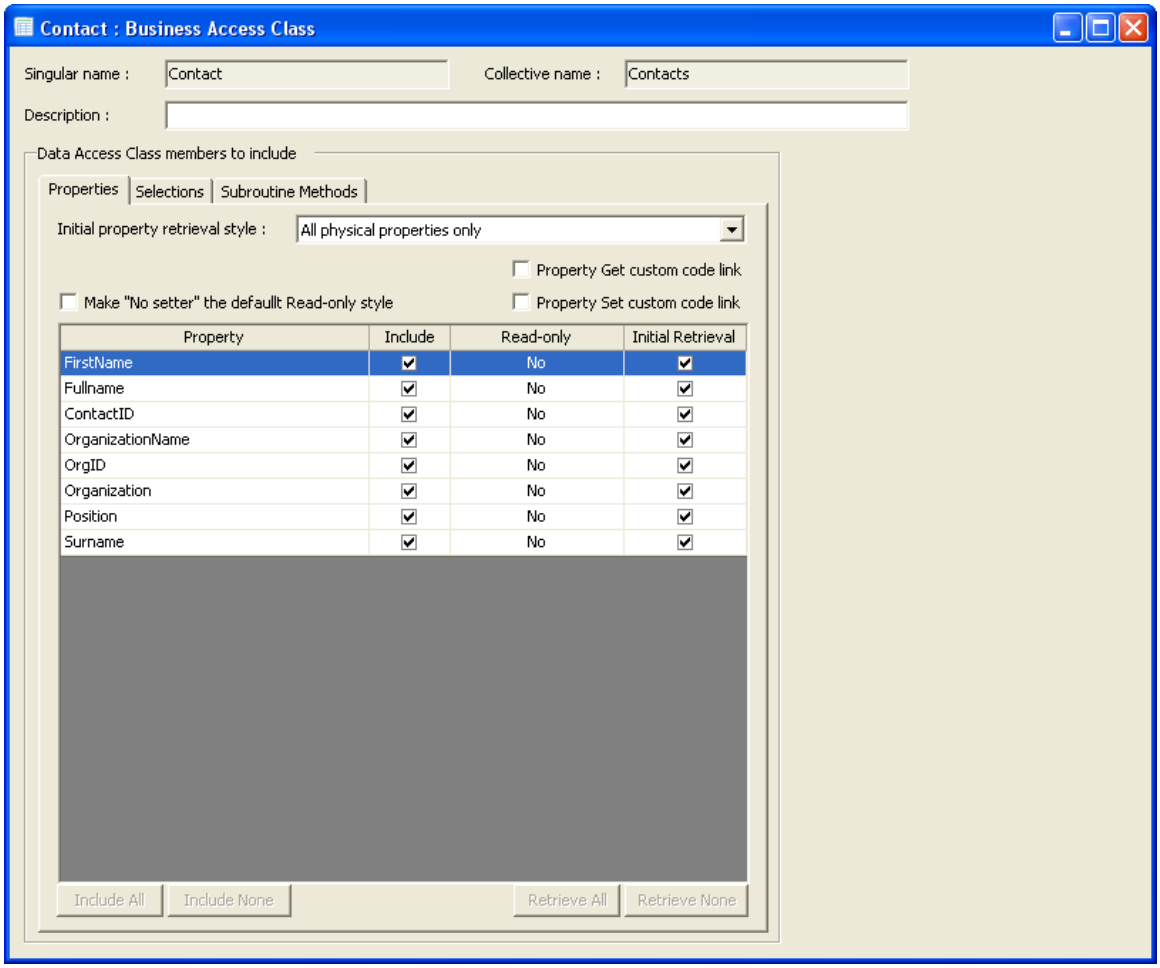
You may create a new BAC by right-clicking the Business Access Classes node and selecting the "Create New Business Access Class" option from the resulting context menu. A dialog prompting you for the singular and collective name of the new BAC will then be displayed.

The 2 input fields contained in the Create New Business Access Class popup dialog box default their content to the singular and collective names of the underlying DAC - you may change them as required.

Clicking the Accept button will result in a new node being created (using the supplied singular name) within the Business Access Classes node.

Maintaining the Definition of an Existing BAC

By double-clicking any of the BAC nodes listed within the Business Access Classes node for an entity you are taken into the BAC maintenance window:



At the top of this form you are able to enter a description for the BAC as a whole. The value of this field will be used in the XML-based IntelliSense help comments within the generated BAL code module.

Beneath this field you are able to maintain the 3 definition groupings of the BAC as described in the following sections.

Defining the Properties of a BAC

The Properties tab of the BAC maintenance window allows you identify which properties of the underlying DAC are to be exposed by the BAC. It also allows you to identify whether each property is to be exposed in a read-only manner and also the property data that is to be retrieved as part of the DAC's initial data retrieval process.

The first (combobox) input field on the Properties tab allows you to identify which category of properties are to be retrieved as part of the DAC's initial data retrieval process.

It is important to understand here that if a property is not included in the list of those that are retrieved as part of the initial data retrieval process it does not necessarily prevent access to that property - it simply means that a further roundtrip to the datasource will be required if any application code references that property.

Clearly, the idea here is that the properties with a high likelihood of being referenced by application code should be included in the list of those that are to be retrieved as part of the initial data retrieval process.

There are 3 choices available in the Initial property retrieval style combobox:

All physical properties only : This option forces all physical (but not calculated) datasource data to be initially retrieved.

All physical properties (physical and non-physical) : This option forces all physical datasource data along with all datasource data relating to calculated base fields referenced by DAC properties to be initially retrieved. Note, in a DAC with many properties based on calculated base fields, this may result in degraded datasource performance.

Specified properties only : This option forces datasource data for base fields (both physical and calculated) referenced by a specified subset of DAC properties to be initially retrieved.

Beneath the Initial property retrieval style combobox are 3 checkboxes, described below:

Make "No setter" the default Read-only style : This option controls which style of property setter is used when the Custom property settings option is unchecked.

Solution Objects supports 2 different ways to implement a read-only property. The first is to flag the property as read-only and to not generate a property setter

clause. The second way is to not flag the property as read-only but to include a empty (no code) property setter.

The reason for including the second option is to ease the task of working with some aspects of .NET data binding. Some .NET controls generate exceptions if a read-only property (i.e. one with no setter clause) is used in a edit context. Thus, the "empty setter" style of read-only property works around this limitation.

Property Get custom code link : This option allows you to indicate whether the Property Get clause within the custom code module is called for properties within this BAC.

There is a very slight overhead incurred in calling into the custom code module for each property value retrieval action. Thus, this option can be unchecked if you are certain that no custom code needs to gain sight of a property value before control is returned to application code.

Please refer to the Integrating Custom Code chapter for more information relating to custom code modules.

Property Set custom code link : This option performs the same function for the Property Set clause as its Property Get counterpart above.

The grid in the lower portion of the Properties tab allows you customize various settings.

The Include column allows you to indicate whether a DAC property is to be included on the interface of this BAC. This column will only be editable if the Custom property settings checkbox is checked.

The Read-only column allows you to indicate the read-only style of a property. This column will only be editable if the Custom property settings checkbox is checked. If the Custom property settings is unchecked, the value for this column for all non-physical properties can be controlled by the use of the 'Make "No setter" the default Read-only style' checkbox.

The Initial Retrieval column allows you to indicate whether the base field data associated with a property is to be retrieved as part of the DAC's initial data retrieval process. This column is only editable if the Initial property retrieval style is set to either "All physical properties and specified non-physical properties" or "Specified properties only". If the value of one of the cells in this column is edited, the Custom property settings checkbox is forced into a checked state.

The Include All/None buttons at the foot of the Properties tab allows you to alter the value of all cells in the Include column of the grid in a single action. These buttons are only enabled if the Custom property settings checkbox is checked.

The Retrieve All/None buttons allow you to alter the value of all cells in the Initial Retrieval column of the grid in a single action. These buttons are only enabled if the Initial property retrieval style is set to either "All physical properties and specified non-physical properties" or "Specified properties only". If the Initial property retrieval style is set to "All physical properties and specified non-physical properties", the Retrieve None button will only uncheck cells for properties associated with calculated base fields.

Defining the Methods of a BAC

The Selections and Subroutine Methods tabs both function in the same manner. They display a grid listing the methods present within the DAC. By default all methods are included. If you wish to omit any methods you must check the "Custom xxx inclusion" checkbox and then use the Include column of the grid to indicate which methods are required for the BAC.

The Include All/None buttons at the foot of the tab are only enabled if the "Custom xxx inclusion" checkbox is checked.

As with properties, if the custom inclusion option is unchecked, any new methods added to the underlying DAC will be automatically including in this BAC. Otherwise, you will need to go back into the BAC maintenance screen and explicitly include them as necessary.

Maintaining Business Access Layer Definitions

This chapter describes how you are able to maintain the definition of the Business Access Layers associated with an Entity .

Overview

The Data Access Layer (DAL) is an implicit concept within Solution Objects in that it is simply the sum total of all DACs defined for an entity. You do not need to do any work to create the DAL other than to define all of the DACs.

The Business Access Layer (BAL), on the other hand, is an explicit concept, that is, it is something which you must explicitly define as part of an entity model definition. Each BAL, ultimately, gets represented as a .NET dll assembly.

A BAL allows you to group together one or more Business Access Classes (BACs) to indicate which classes need to be present within the generated code that will eventually manifest the BAL.

You may define any number of BALs. You must, however, define at least one. Each BAL has its own generated code module produced by the code generation process.

The concept of a BAL is based around the idea of being able to create a grouping of BACs that is strongly focused on providing a certain community of application developers with a set of classes tuned to their specific requirements. This set of classes must, obviously, provide the level of functionality that they require in order to perform their tasks but it may well be possible to define specific BACs that

exclude certain non-required functionality in order to reduce the complexity or size of a class' interface. Specific BACs may also be produced to hide sensitive or confidential property data.

Thus, the concept of a BAL works closely with the concept of a BAC. A BAC may be included in any number of BALs.

Creating a New BAL

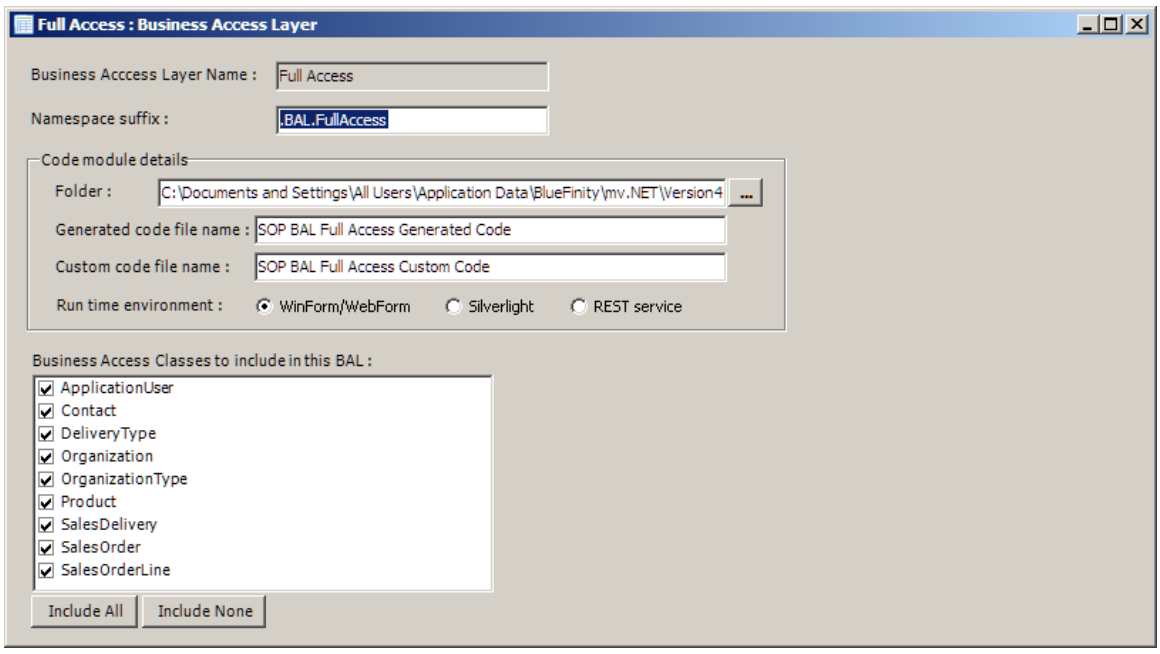
Each entity model version within the Data Manager's treeview has a sub-node named "Business Access Layers". It is within this node that the BALs for an entity are listed.

You may create a new BAL by right-clicking the Business Access Layers node and selecting the "Create New Business Access Layer" option from the resulting context menu. A dialog prompting you for the name of the new BAL will then be displayed.

Clicking the Accept button will result in a new node being created (using the supplied name) within the Business Access Layers node.

Maintaining the Definition of an Existing BAL

By double-clicking any of the BAL nodes listed within the Business Access Layers node for an entity you are taken into the BAL maintenance window:



The first input field on this form allows you to enter the namespace setting that will be appended to the root namespace of the entity model as a whole.

Next, you are able to enter the location and names of the 2 code modules that will be produced for the BAL by the code generation process. Note, you do not enter language extensions at the end of code file names (e.g. .vb or .cs) - these extensions will be added automatically by the code generation process depending on the language selected at generation time.

The Run time environment options allow you to specify which run-time environment a BAL is targeted at. The code generator adjusts its output according to which option is selected for a BAL.

The "Silverlight" option should be selected if the assembly generated from this BAL is going to be used within the browser-resident project of a Silverlight application. Please refer to the Silverlight Developer guide for more details on this topic.

The “REST service” option should be selected if the generated assembly is going to be used by an ASP.NET hosted REST service. Please refer to the REST Service Developer Guide for more details on this topic.

For all other run-time environments, the “WinFom/WebForm” option should be selected.

In the lower half of the screen you are able to specify which BACs within the entity model are to be included in this particular BAL.

Validating Entity Definitions

This chapter describes how you are able to validate that your entity model definitions are consistent with both the underlying datasource schema and with each other.

Overview

There is a certain amount of validation performed when you save a DAC definition. However, this validation is not able to detect all potential problems. For this reason, there are 2 places within the Data Manager where you are able to perform explicit validation of all or part of an entity model:

- Validating a single DAC
- Validating the entity model as a whole

Validating a Single DAC

The right-click context menu of a Data Access Class node contains a menu option called "Validate Data Access Class Definition". This option allows the details of the associated DAC to be checked for a number of potential issues.

Once this option is chosen, the definition of the DAC is read and the following checks performed:

- Comparison of the DAC definition with the current schema of the underlying datasource.
- Validation of various internal keys within the DAC definition.
- Validation of datasource name references within the DAC definition.

Any problems encountered are listed in a grid on the resulting form. The details of each problem is shown along with a description of the resolution required to remove the problem. Some problems require you to select an alternative value, in which case the right-most column of the grid will present the list of available options from which you may select the required replacement value.

Validating the Whole Entity Model

The right-click context menu of a Version node contains a menu option called "Validate Model". This option allows the definition details of an entire entity model version to be validated.

Once this option is chosen, the definition of the version is read and any validation errors are displayed within a grid. This grid works in the same way as the DAC validation form described above.

Other than the fact that this validation form validates an entire version, the other main difference with the DAC validation window is that the version validation window is modal, which mean that you are able to adjust definition details using other windows without have to close down the version validation form. Hence, there is a Re-Validate button at the bottom of the form which allows you to re-run the validation actions on the current entity model definitions. Any changes to underlying file schema are also recognized by the Re-Validate action.

The version validation window has a "Generate Code Modules" button at the foot of the form that becomes enabled when no validation errors exist. This invokes the standard code generator window as described in the following Generating Code Modules chapter.

Generating Code Modules

This chapter describes how you are able to use your entity definitions to generate a series of source code modules.

Overview

Generating source code is the raison d'être of the Solution Objects functionality within the Data Manager.

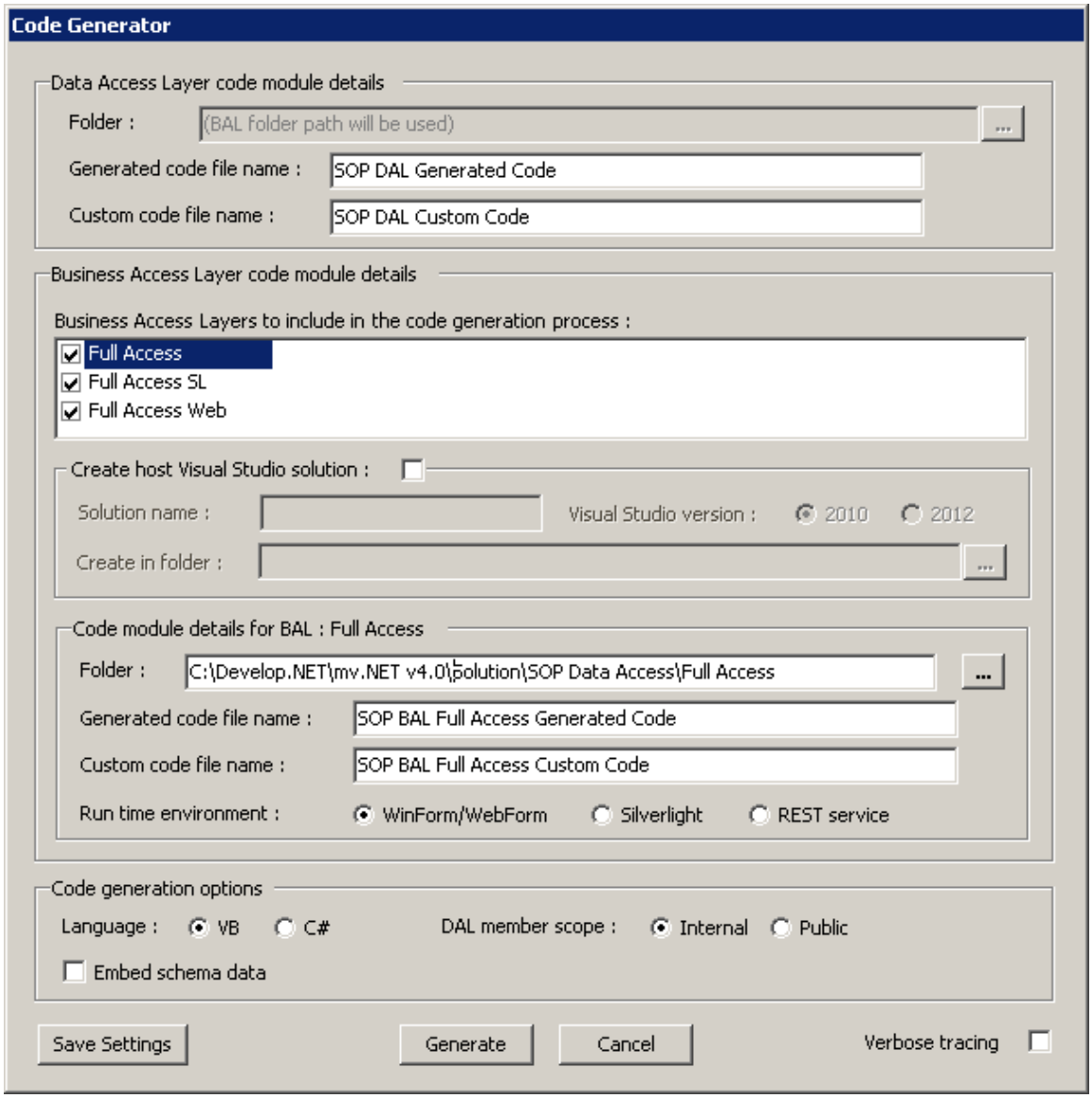
Once you have created at least one DAC, one BAC and one BAL within your entity model, you are ready to roll! Of course, you will almost certainly have many of these in your entity definition.

Invoking the Code Generator

Code generation is performed at the entity model version level. Thus, you access the code generator by right-clicking the relevant version node within your entity model and selecting the "Generate Code Modules" option from the resulting context menu.

The code generator can also be invoked from within the entity model validation screen. Please refer to the previous chapter for further details on this topic.

Once the code generator has been invoked, the following window will be displayed:



The top groupbox area of this screen allows you to enter the location and names of the 2 code modules that will be produced for the DAL by the code generation process. Note, you do not enter language extensions at the end of code file names (e.g. .vb or .cs) – these extensions will be added automatically by the code generation process depending on the language selected at the foot of this screen.

The middle section of the screen allows you to select which BALs you wish to include in this particular code generation action and to also enter the location and

names of the 2 code modules that will be produced for each BAL. The settings for each BAL are drawn from the information previously entered within the BAL maintenance window.

This center section also allows you to generate an entire Visual Studio solution to host the generated code if you do not already have one created.

At the foot of the screen you are able to set a number of options that control the content of the generated code. You able to:

- Choose the language that will be used for the code generation action.
- Specify the scope that all DAL interface members will be set to. If you are going to combine the DAL code modules with the BAL code modules in a single Visual Studio project you are best selecting the "Internal" scope option. This will result in the DAL classes being excluded from the public interface of the generated assembly.

  If you are going to create a separate assembly just for the DAL you will need to select the "Public" scope option.
- Indicate whether you would like the schema (dictionary) data of the files upon which your entities have been based to be embedded within the generated code. This will increase the size of your assembly but will improve application performance.

Finally, as the code generator processes the entity model, is produces a trace of its progress. If errors are encountered you may wish or be requested by BlueFinity support to activate the Verbose tracing option in the bottom right of the form.

Clicking the Generate button will first save the settings in this screen and will then initiate the code generation process. The code generator displays its progress throughout the generation action.

When the code generation is completed, any errors encountered are displayed.

The Save Settings button in the bottom left corner of the screen allows you to save the details of the generator form without actually running the generation process.

Using Generated Code

This chapter describes how you are able to use the code that Solution Objects generates to create one or more access layer assemblies.

Overview

Once you have generated your code modules, the final step in producing an access layer that can be distributed to your community of application developers is to, using Visual Studio, create an assembly incorporating the generated code.

Note, in the following sections reference is made to the "Program Files" folder, if you are running on a 64 bit system this will be "Program Files (x86)".

Steps to Produce an Access Layer Assembly

The easiest way to create the Visual Studio project(s) to host your BAL code is to let the code generator do it for you (see previous chapter). Alternatively you can do it all manually as described below:

1. Launch Visual Studio.
2. Create a new Class Library project, selecting the language that matches the one you selected when you generated the code within the Data Manager.
3. Go into the Project Properties window and set the required assembly name for your project. For VB.NET projects also clear out the Root/Default namespace field in this form.

4. Add a reference to the required assemblies.

   For BALs targeting a non-Silverlight runtime environment, add a reference to the following 2 assemblies:

   ```
   Program Files\BlueFinity\mv.NET\Version4.0\bin\Public Assemblies\
   BlueFinity.mvNET.SolutionObjects.dll
   ```

   ```
   Program Files\BlueFinity\mv.NET\Version4.0\bin\
   BlueFinity.mvNET.CoreObjectsDAL.dll
   ```

   For BALs targeting a Silverlight runtime environment, add a reference to all assemblies in the following folder:

   ```
   Program Files\BlueFinity\mv.NET\Version4.0\bin\Silverlight
   ```

   Additionally for Silverlight projects, you will need to add a reference to the following assembly:

   ```
   Program Files\Microsoft SDKs\Silverlight\v4.0\Libraries\Client\
   System.Windows.Data.dll
   ```

5. Right-click the project node within the Visual Studio Solution Explorer window and select the "Add -> Existing Item" menu option. Browse to the location of your generated code modules and add them to your project.

6. Select the project build option to create your assembly.

If your DAL is to be contained in a separate assembly to that of your BAL, you will need to repeat the above process for your BAL, except that instead of referencing the `BlueFinity.mvNET.CoreObjectsDAL.dll` assembly you will need to reference your DAL assembly.

Utilizing a Business Access Layer

This chapter describes how you are able to use the business access layer assembly that you have created using the steps in the previous chapter. It also contains a summary of the properties and methods that are automatically included on the interface of a BAC in addition the properties and methods defined in the entity model definition.

Overview

Once you have created your business access layer assembly all that you need to do is add a reference to it within your Visual Studio application project(s).

Steps for Utilizing a Business Access Layer Assembly

1. Launch Visual Studio for your application project.
2. Add a reference to the following assembly:

   ```
   Program Files\BlueFinity\mv.NET\Version4.0\bin\Public Assemblies\
   BlueFinity.mvNET.SolutionObjects.dll
   ```

3. Add a reference to the BAL assembly that you have created in Visual Studio as per the previous chapter.

4. Optionally, you may add Imports/using statements at the start of your application code modules for the above assemblies to shortcut the use of namespaces.

BAC Standard Properties and Methods

The code generator will automatically include a number of standard properties and methods on the interface of the singular class of each BAC. These properties and methods are summarized below and are covered in greater depth within the following chapters of this guide.

Name	Type	Description
Create	Method	Allows new instances of the class to be created within the underlying data store. See associated chapter below.
Delete	Method	Allows instances of the class to be deleted from the underlying data store. See associated chapter below.
Exists	Method	Allows you to test whether an instance already exists within the underlying data store.
PrimaryKey	Property	A string version of the Item ID/Primary key of the instance.
SelectAll	Method	Selects all instances of the class within the underlying data store. This method is only included if the "Include SelectAll" checkbox within the Data Access Class maintenance window is ticked.
SelectGeneric	Method	Selects a range of instances of the class within the underlying data store based on database-specific selection and sort criteria. This method is only included if the "Include SelectGeneric" checkbox within the Data Access Class maintenance window is ticked.
StaticData	Property	An EntityStaticDataBAL instance holding a range of static data values for the class. See below for further details.
Update	Method	Allows property value amendments to be persisted within the underlying data store. See associated chapter below.
<web data binding methods>	Methods	Please refer to the WebForm Data Binding Support chapter for details on these methods.

Class Static Data

The code generated for a BAC will automatically include a property called "StaticData" on the singular class. The value returned by this property is an instance of the EntityStaticDataBAL class which is a specialized class designed to hold several pieces of class-level data.

The values held by the EntityStaticDataBAL instance are set at class initialization time and are not, generally speaking, intended for programmer use. However, in certain circumstances it may be desirable to amend the values held in some of the properties of this EntityStaticDataBAL instance. Below is a summary of the properties of the EntityStaticDataBAL class which are intended for end-programmer usage (if required).

Property Name	Description
PhysicalProperties	The names (space separated list) of the physical properties to be retrieved in the initial data retrieval action.
CalculatedProperties	The names (space separated list) of the non-physical properties to be retrieved in the initial data retrieval action.
AutoIDPreEmptiveSize	The number of auto generated IDs to be pre-emptively retrieved when a new ID is requested. This value allows auto ID generation to be optimized when an auto ID value is required at the time of creating a new instance (as opposed to when a new instance is saved to the data store). By retrieving a set of auto IDs, the number of database round-trips can be reduced when a number of instances are going to be created. The downside of this is that some of the pre-emptively retrieved IDs by never be used.
AutoIDPreEmptiveSizeReset	Turns off pre-emptive auto ID retrieval.
StaticDataDAL	The associated EntityStaticDataDAL instance. See below.

The StaticDataDAL property holds an instance of the associated EntityStaticDataDAL class. This class holds various pieces of class-level data for the underlying DAC associated with the BAC. As with the EntityStaticDataBAL instance held at BAC level, the EntityStaticDataDAL instance held at DAC level is

not, generally speaking, intended for programmer use. However, in certain circumstances it may be desirable to amend the values held in some of the properties of this EntityStaticDataDAL instance. Below is a summary of the properties of the EntityStaticDataDAL class which are intended for end-programmer usage (if required).

Property Name	Description
DataFileName	The name of the database file holding the persisted data of the class.
SchemaFileName	The name of the database file holding the schema associated with the class.

If may be necessary to maintain the values of the above 2 properties if a single entity definition is being used to access multiple database files all of the same structure.

Integrating Custom Code

This chapter describes how you are able to integrate the program code within a custom code module with the code generated by Solution Objects. It also covers how to override the default error messages produced by Solution Objects.

Overview

The code generated by Solution Objects consists of class declaration and implementation code. All of these classes are declared as partial classes, the intention being that you are then able to extend the features of any class by creating code in a separate file. The .NET compiler will then join these 2 file modules (the Solution Objects generated code and your custom code) together to create a single consolidated class definition.

The Custom Code File

The Solution Objects code generator produces 2 code modules per DAL/BAL. The first file (referred to as the "generated code file") should be regarded as source code which should not be amended by you the developer under any circumstances – this is because the code generator removes and regenerates the entire contents of this file each time the code generation process is performed.

The second generated file (referred to as the "custom code file") is deemed to be "owned" by you. The only times that the code generator will alter the contents of this file are:

1. The very first time code is generated for an entity model.

2. When the code generator detects that a class declaration is not present within the custom code file. In which case, it will add a new class declaration "stub" to the custom code file.

The purpose of the custom code file is to provide a place where you are able to add members to the interface of a class and to intercept the occurrence of the following events:

1. A property value Get and Set
2. The invocation of a CRUD action (before and after)

The interception of property value Get and Set is available for both the DAL and BAL. The interception of CRUD actions is only available for the DAL.

Intercepting Property Get/Set in the DAL

Within the DAL custom code module, each class stub generated by Solution Objects contains a PropertyGet and a PropertySet method.

The PropertyGet method allows you to link your own code into the property Get process for one or more properties in the class. The PropertyGet method stub generated by Solution Objects is shown below:

```
Friend Function PropertyGet(ByVal PropertyEnum As String, ByVal PropertyValue
As Object) As Object

    Select Case PropertyEnum

    End Select

    Return PropertyValue

End Function
```

The Select clause within the body of the function allows you to perform a specific block of code for each property within the class. In order to allow you to identify properties more easily here, the code generator creates a series a static variables within a namespace called *{classname}*Property, e.g. OrganizationProperty. Below is a snippet of code which illustrates the use of this (the assumption here is that the Organization class has a property called "Name":

```
Select Case PropertyEnum
    Case OrganizationProperty.Name
        Return PropertyValue.Substring(0, 1).ToUpper &
                                        PropertyValue.Substring(1).ToLower
End Select
```

The above code forces the value returned by the Name property to follow a predefined upper/lower casing specification.

The DAC maintenance form allows you to define whether the PropertyGet function is called before and/or after the property value is extracted from the underlying datasource record (see section on defining property values more details).

If it is defined that the PropertyGet method is to be called before the base record data is extracted (i.e. the "DAL custom code Get Pre-check" checkbox is ticked within the DAC maintenance form), the PropertyValue argument will be set to null/nothing before a call into the PropertyGet method is performed. If you return anything other than a null/nothing value, that value will be used as the property value and no data will be extracted out of the base record for that property.

If it is defined that the PropertyGet method is to be called after the base record data is extracted (i.e. the "Get Post-check" checkbox is ticked within the DAC maintenance form), the PropertyValue argument will be set to the data extracted from the base record. Again, if you return anything other than a null/nothing value, that value will be used as the property value.

The PropertySet method allows you to link your own code into the property Set process for one or more properties in the class. The PropertySet method stub generated by Solution Objects is shown below:

```
Friend Function PropertySet(ByVal PropertyEnum As String, ByRef PropertyValue
As Object) As String

    ' Set return value to a non-empty string to indicate an error

    Select Case PropertyEnum

    End Select

Return ""

End Function
```

Note, the PropertySet method is always called as part of the property value setting process. The PropertyValue argument is set to the candidate value for the property. You may adjust this as necessary. If you need to block the value from being set into the base record, you need to throw an exception from within your custom code.

Intercepting CRUD actions in the DAL

Within the DAL custom code module, each class stub generated by Solution Objects contains the following 2 methods:

BeforeCRUD

AfterCRUD

These methods are designed to be used when you need to invoke custom logic before and/or after a specific CRUD operation.

BeforeCRUD Code Stub

The code stub inserted here (shown in VB for entity type "Contact") is as follows:

```
Friend Shared Function BeforeCRUD(ByVal CRUDType As CRUDType, ByVal Instance
                                                        As Contact) As String

    ' Set return value to a non-empty string to indicate an error

    Dim returnValue As String = ""

    Select Case CRUDType
        Case CRUDType.Create

        Case CRUDType.Read

        Case CRUDType.Update

        Case CRUDType.Delete

        Case CRUDType.Select

    End Select

    Return returnValue

End Function
```

If you wish to abort the CRUD action the return value should be set to a non-empty string. Note, the Solution Objects framework will not raise an exception if a string value is returned - you must do this with your custom code if you require this to be done.

The second argument of the BeforeCRUD method is only supplied during the Update or Select actions - for all other actions it is set to null.

For Update actions it contains the instance of the entity that is about to be updated.

For Select actions it contains an empty instance of the relevant DAC and is provided purely to allow programmatic access to the arguments that have been supplied in the invocation of the selection method. Access to selection method arguments is via

the DAC's SelectionMethodCallArgs property – this property returns an instance of the SelectionMethodCallArgs class which contains the following properties:

```
SelectionMethodName
ArgumentValues
```

The SelectionMethodName property holds the name of the selection method being invoked. The ArgumentValues property holds a Dictionary(Of String, Object) collection variable which gives you access to the selection method arguments (based on argument name. Any changes to the content of this Dictionary collection will be picked up and used in the ensuing selection action.

For example, the code below forces the "VersionID" argument of the "SelectByVersion" selection method to a value of "3":

```
Case CRUDType.Select
    If Instance.SelectionMethodCallArgs.SelectionMethodName =
                                                    "SelectByVersion" Then
       Instance.SelectionMethodCallArgs.ArgumentValues("VersionID") = 3
    End If
```

AfterCRUD Code Stub

The code stub inserted here (shown in VB for entity type "Contact") is as follows:

```
Friend Shared Sub AfterCRUD(ByVal CRUDType As CRUDType, ByVal Instance As
                                                                Contact)

    Select Case CRUDType
        Case CRUDType.Create

        Case CRUDType.Read

        Case CRUDType.Update

        Case CRUDType.Delete

    End Select

End Sub
```

The second argument of the AfterCRUD method is only supplied during the Read action and contains the instance of the entity that has just been read.

Note, the AfterCRUD method is not invoked for the Select CRUD type.

Overriding Error Messages

The code generated by solution objects (along with the support assemblies used at run-time) will generate exceptions under certain error conditions. By default, the description of these exceptions will be in English but you are able to replace these default descriptions with your own language or context dependent versions.

In order to do this you will need to perform the following actions:

1. At the start of your application you will need to assign a language code (of your choosing) to the "LiteralStrings.LanguageCode" variable. The LiteralStrings class will be within your DAL namespace, for example:

   ```
   BlueFinity.SOP.DAL.LiteralStrings.LanguageCode = "ESN"
   ```

2. Add your custom error descriptions into the SO DAL code template. The DAL code templates are held in the following folder:

   `C:\Program Files\BlueFinity\mv.NET\Version4.0\Code Templates\DAL`

   For VB, the code template file is called "`DAL VB.txt`", for C# the code template file is called "`DAL CS.txt`"

   Within the template file search for the line containing the string "{Insert custom literal strings here}". At this point you will need to add overrides for all of the SO messages as required. An example is given below:

   VB

   ```
   Public Overrides ReadOnly Property INVALID_Int32_VALUE As String
       Get
           Select Case LiteralStrings.LanguageCode
               Case "ESN"
                   Return "El incorrecto valor de integer número entró"
               Case Else
                   Return MyBase.INVALID_Int32_VALUE
           End Select
       End Get
   End Property
   ```

   C#

   ```
   public override string INVALID_Int32_VALUE
   {
        get
        {
                switch (LiteralStrings.LanguageCode)
                {
                        case "ESN":
                                return "El incorrecto valor de integer número entró";
                        default:
                                return base.INVALID_Int32_VALUE;
                }
        }

   }
   ```

Solution Objects uses the following error message literal strings:

Literal String ID	Default (English) Value
COLLECTIVE_INSTANCE_NOT_FOUND	Item not found in the collection
COLLECTIVE_ASSIGN_INVALID	Cannot assign directly into collective instance
CANNOT_ASSIGN_UNKNOWN_ENUM	'UnknownValue' enumeration value cannot be assigned
CONNECTION_OPEN_FAILED	Open of database connection failed
DATABASE_ERROR	Database error
DATASOURCE_NOT_FOUND	Datasource not found within repository
DATA_IS_INVALID	data contains invalid property values
EMPTY_VALUE_NOTALLOWED	Empty/null value cannot be assigned to this property
FK_DATASOURCE_NOT_DEFINED	Foreign key datasource not defined
FK_DATASOURCE_NOT_FOUND	Foreign datasource file not found
FK_NOT_FOUND	Foreign datasource value not found in associated datasource
INVALID_Boolean_VALUE	Invalid true/false value entered
INVALID_DATE	Invalid date value
INVALID_DateTime_VALUE	Invalid date/time value entered
INVALID_Double_VALUE	Invalid numeric value entered
INVALID_Decimal_VALUE	Invalid numeric value entered
INVALID_KEY_LIST	Invalid primary key list supplied
INVALID_Int32_VALUE	Invalid integer value entered
INVALID_Int64_VALUE	Invalid integer value entered
INVALID_PROPERTY_NAME	Property name not known
ITEM_MODIFIED_BY_ANOTHER_USER	The item has been modified by another user
ITEM_LOCKED_BY_ANOTHER_USER	The item is currently locked by another user
ITEM_ALREADY_EXISTS	The item already exists
MVORDINAL_NOT_PRESENT	Mv ordinal position property not present in collection

PESSIMISTIC_LOCKING_NOT_ALLOWED	Pessimistic locking is allowed on nested entities
PK_CANNOT_BE_BLANK	Primary key value cannot be blank
PK_CANNOT_BE_ALTERED	Existing primary key value cannot be altered
PK_USES_AUTOID	Primary key value is to be assigned via auto ID generation
RECORD_NOT_FOUND	Record not found
SVORDINAL_NOT_PRESENT	Sv ordinal position property not present in collection
TRANSACTION_ALREADY_IN_PROGRESS	Unable to start a new transaction - one is already in progress
TRANSACTION_COMMIT_FAILED	transaction(s) failed to commit
TRANSACTION_NOT_IN_PROGRESS	"A transaction is not currently in progress"
UPDATE_ERROR	Update error(s) encountered
USE_CREATE_AND_INSERT	Use Create and Insert methods for non-nested data insertion

Reading/Selecting Data

This chapter describes how you are able to use your generated Business Access Layer to read and select data from the underlying datasource.

Overview

The first thing that you'll probably want to do with your BAL is to use it to read information from your database and have this data presented in the form of instances of the various classes within your BAL. In order to do this you will need to use one or more of the properties and methods created as part of the code generation phase.

The following sections of this chapter describe the various ways in which you can do this.

Initializing Data Access

All of the static methods created by the code generator require a "DataRepository" instance to be passed into each method in order for Solution Objects to know where the underlying application datasource can be found.

A "DataRepository" is an instance of the DataRepository class within the Solution Objects runtime assembly. The following code snippet illustrates how a DataRepository instance can be created:

VB:

```
Dim repository As BlueFinity.mvNET.SolutionObjects.DataRepository =
CompanyABC.OrderProcessing.BAL.FullAccess.Repository.Initialize()
```

C#:

```
BlueFinity.mvNET.SolutionObjects.DataRepository repository =
CompanyABC.OrderProcessing.BAL.FullAccess.Repository.Initialize();
```

As can be seen above, each BAL contains a "Repository" class. This class has a static method named "Initialize" which is a factory method for creating DataRepository instances for use with that specific BAL.

The Initialize method overload used in the above code will use the default datasource connection string. This is the connection string defined within the entity model. The Initialize method also has an overload which allows you to supply the connection string at run time. The format of the connection string will depend on the type of underlying datasource. For a MultiValue datasource, the format of the connection string is as follows:

```
Server={LoginProfileName}
```

Where *{LoginProfileName}* is the required login profile as defined within the mv.NET Data Manager utility.

e.g.

VB:

```
Dim SOPdata As BlueFinity.mvNET.SolutionObjects.DataRepository =
CompanyABC.OrderProcessing.BAL.FullAccess.Repository.Initialize("Server=SOP")
```

C#:

```
BlueFinity.mvNET.SolutionObjects.DataRepository SOPdata =
CompanyABC.OrderProcessing.BAL.FullAccess.Repository.Initialize("Server=SOP");
```

There are some optional extra details that can be supplied within the connection string, each of which is delimited by a pipe ("|") character:

```
ApplicationID={ApplicationID}|ClientGUID={ClientGUID}|User={UserID}|Password={Passwo
rd}|SessionManagerAddress={SMAddress}
```

Where:

{ApplicationID} = a description of the application using the repository.

{ClientGUID} = A unique string identifying a specific instance of a client. This needs to be supplied if you wish to maintain collective instances of entities across DataRepository instances. For example, in web applications you would typically set this value on first use of a DataRepository within a web page and then persist its value in ViewState or some other persistence mechanism in order for it to be available for use in subsequent code-behind executions. The GUID value is used to name datasource hosted resources associated with collective instances.

{UserID} = A string identifying the user name to be used in the datasource connection/authentication process

{Password} = The password to be used in the datasource connection/authentication process

{SMAddress} = The address of the mv.NET Session Manager. This is only relevant for MultiValue datasources

Reading Individual Entity Instances

Each singular BAC has a static "Read" method which allows you to instantiate individual entity instances using data from the underlying datasource.

e.g.

VB:

```
Dim myOrg As Organization = Organization.Read(SOPdata, txtOrg.Text)
```

C#:

```
Organization myOrg = Organization.Read(SOPdata, txtOrg.Text);
```

The Read method has 3 overloads as indicated in the grid below (the example above uses overload#1):

Argument	Used in Overload #	Description
Repository	1,2 & 3	The DataRepository instance to be used
PrimaryKey	1,2 & 3	The primary key of the required datasource instance
LockStyle	2 & 3	The locking style (within the underlying datasource) to be used. See below for further details on this.

CreateIfNew	3	Indicates whether a new instance is to be created if the supplied primary key does not exist

The LockStyle argument, if supplied, indicates which method of locking within the underlying datasource level is to be used by the read operation. It may be set to one of the following enumeration values:

Enum Name	Description
None	No record locking is to be used.
Optimistic	Optimistic record locking is to be used. This setting will always allow the read operation to be performed but will check for update contention on update commit.
Pessimistic	Pessimistic record locking is to be used. This setting will only allow the read operation to be performed if no other user currently has a pessimistic lock held on the same underlying datasource record. The pessimistic lock will be released on update commit (unless lock retention is specified - see next chapter).
PessimisticFallback	Pessimistic record locking is to be used. If another user currently holds a pessimistic lock on the same underlying datasource record, the read will still be performed but no record locking will be applied. You may interrogate the "IsLocked" property of the entity instance to find out whether a pessimistic lock has been successfully obtained.
PessimisticOnModify	Pessimistic record locking will be used on first property value amendment. A non-record locking read will be used initially. If another user currently holds a pessimistic lock on the same underlying datasource record at the time when the pessimistic lock is attempted, an exception will be raised to the end application. The pessimistic lock (if obtained) will be released on update commit (unless lock retention is specified - see next chapter).
PessimisticOnModifyFallback	This is a combination of PessimisticFallback and PessimisticOnModify in that the lock will be applied on first property value amendment but if that fails,

	no exception will be raised.

Selecting Multiple Entity Instances

For each self (static) selection method identified as being required on the interface of a BAC there will be a corresponding static method on the singular class of the BAC. These static methods will each have a number of overloads providing you with a certain degree of control over how the selection is performed. The following grid summarizes each argument that may appear on these overloads:

Argument	Description
{specific arguments}	Selection method specific arguments. There may be zero or more of these for each selection method.
Repository	The DataRepository instance to be used
PrimaryKey	The primary key of the required datasource instance
LockStyle	The locking style to be used on each selected datasource record. Values of None, Optimistic and Pessimistic are allowed here - note, any of the Pessimistic locking style variants will be treated as simply Pessimistic. See above for more details. The default value here is None.
SelectionProfile	This argument can be set to one of two possible enumeration values: FetchOnDemandAndServerPersistanceOff FetchOnDemandAndServerPersistanceOn If the "Off" setting is chosen, all selection data will be retrieved in a single round-trip to the underlying datasource. No state persistence information will be created on the datasource and no paging will be supported. If the "On" setting is specified, underlying datasource data will be retrieved on demand (this is used by the in-built paging mechanism) and state information will be created within the underlying datasource. This state information allows collective instances to be recreated without re-executing selection commands on the underlying datasource. The number of records retrieved when the on-demand mechanism is activated is controlled by the

	page size specified in the PageSize argument (see below). The default setting for this argument is controlled by the "DefaultSelectionProfile" property of the DataRepository instance. This property itself defaults to a value of FetchOnDemandAndServerPersistanceOn.
AdditionalProperties	A list of property names (space separated) indicating additional properties (i.e. additional to those specified in the BAC definition) to be retrieved in the initial data retrieval phase. This allows an end-application to optimize datasource retrieval based on the data needs of a specific section of code within the application.
PageRowOffset	The ordinal index (zero-based) of the row within the overall list of selected records to be used as the first row within the returned collection.
PageSize	The number of rows to be extracted (starting at the PageRowOffset row) from the overall list of selected records to form the returned collection.
SortOverride	The property to be used to sort the returned collection. This will override the sort specification of the underlying selection method.

Pre-emptive Data Selection

All collective classes support a mechanism to allow you to retrieve the data associated with a singular object reference property for all instances in the collection in a single data source roundtrip. This is perhaps best explained using an example.

Let us say that we have an Organization class. This Organization class has a string property called MainContactID which holds the primary key of a Contact class instance. The Organization class also has a property called MainContact which is an object reference property, that is, it holds an instance of a Contact object – the instance indicated by the content of the MainContactID property.

If we use one of the selection methods of the Organization class to obtain an Organizations collective instance, the individual Organization instances within that collection will not yet hold an object reference in the MainContact property – i.e. it will be null/Nothing. If we were to process each Organization instance in a loop, referencing the MainContact property within the loop, we would trigger a

roundtrip to the data source for each loop iteration. This obviously works, but it is not particularly efficient – especially if we have many entries in the collection.

Therefore, all collective classes support a method called LoadEntityPropertyData. This method allows you to specify the name of an object reference property (an enumeration allowing easy identification of a property is automatically created by the code generator) and will result in the data for all instances of the related class referenced by the entries in the collection being retrieved in a single data source roundtrip.

Saving Data Changes

This chapter describes how you are able to save the changes that you make to an entity instance's property values to the underlying datasource.

Overview

Each (singular) Business Access Class (BAC) has an Update method. This method has a number of overloads allowing you to control the precise behaviour of data updating.

There is also an instance-based Update method on the collective class of each BAC that can be used to save all modified entries in a collection of BAC instances.

Each BAC also has static (class-level) Update method that is included as part of Solution Objects' Web data binding support. Please refer to the WebForm Data Binding Support chapter for further details.

Singular Instance Update Method

If you hold a singular BAC instance within your application, you may save its current data content by using its "Update" instance method. This method has the following overloads:

```
Update()

Update(ByVal CascadeUpdate As Boolean, ByVal RetainLock As Boolean)

Update(ByVal OrgID As String)
```

The first overload simply calls the second overload passing in argument values of True and False respectively.

The second overload saves the current data content of the BAC if any of its property data values have been modified. If the CascadeUpdate argument is set to True, the update process will scan the properties of the BAC instance for the presence of any singular or collective BAC instances and will invoke the Update instance method for any that are found. In this way, only a single invocation of the Update method (of the top-level BAC) is required to save a logical "tree" of related BAC instances. If the RetainLock argument is set to False and pessimistic locking has been used, the pessimistic lock with be released, otherwise it will be retained.

The third overload allows the data to be saved under a different Item ID (primary key).

Collective Instance Update Method

If you hold a collective BAC instance within your application, you may save the current data content of all modified BAC instances contained within the collection by using its Update instance method. This method has the following signature:

```
Sub Update(ByVal CascadeUpdates As Boolean)
```

This method scans the collection for BAC instances holding unsaved data modifications and for any found invokes their individual Update method.

If the CascadeUpdates argument is set to True, the update process will scan the properties of each BAC instance within the collection for the presence of any singular or collective BAC instances and will invoke the Update instance method for any that are found. In this way, only a single invocation of the Update method (of the top-level BAC collection) is required to save a logical "tree" of related BAC instances

Deleting Data

This chapter describes how you are able to delete entity instance's from the underlying data store.

Overview

Each (singular) Business Access Class (BAC) has a static (class-level) Delete method. This allows you delete individual records from the underlying data store.

There is also an instance-based Delete method on both the singular and collective class of each BAC which can be used to delete one or more BAC instances.

Additionally, each (singular) BAC also has a static (class-level) Delete method that is included as part of Solution Objects' Web data binding support. Please refer to the WebForm Data Binding Support chapter for further details.

Static Delete Method

Each BAC has a "Delete" static method on the singular class. This method allows you to specify the item ID (primary key) of the record in the underlying data store to be deleted.

Instance Delete Method

Each BAC has a "Delete" instance method on the singular class. This method allows you to delete the instance of the class in the underlying data store. The instance Delete method has an overload which allows you to control 2 aspects of deletion:

```
Public Sub Delete(ByVal CascadeDelete As Boolean, ByVal RetainLock As Boolean)
```

The "CascadeDelete" argument allows you to indicate whether cascaded delete processing should be performed. Cascaded delete processing is a mechanism whereby all members of the interface of a class which are flagged as supporting cascaded deletes will be deleted in addition the base instance.

For example, if we take the Organization class from the demo SOP entity model it has a property which holds the primary key of the main contact person for that organization. There is also a property which supplies the Contact class object instance representing that primary key value. The Contact class property can be flagged (via the Data Access Class Maintenance window) as supporting deleted cascading. Therefore, when an Organization instance is deleted and the Contact class property is flagged as supporting deleted cascading, the main contact record will also be deleted from the underlying data store.

The "RetainLock" argument of the Delete instance method allows you to indicate whether a pessimistic lock within the underlying data store on the item being deleted is to be retained.

Creating New Instances

This chapter describes how you are able to create new entity instance's in the underlying data store.

Overview

Each (singular) Business Access Class supports a number of ways to create new instances of the associated entity. The method you use depends on a number of factors:

- Does the underlying datasource file use auto ID generation?
- Do you know the primary key of the new instance at the time of initial creation?

Static Create Method

Each BAC has a "Create" static method on the singular class. This method creates a new instance of the relevant class in memory only. Only when the Update method of the new instance is used will this instance be stored in the underlying data store.

The Create method has a call signature which will vary depending on whether the underlying datasource file uses auto ID generation.

If the underlying datasource file uses auto ID generation, the Create method will have the following call signature:

```
Public Shared Function Create(ByVal Repository As DataRepository, ByVal
                WithImmediateAutoID As Boolean, ByVal LockStyle As LockingStyle) As Contact
```

The "WithImmediateAutoID" argument allows you to control whether the auto ID generation occurs at the time when the Create method is invoked (i.e. WithImmediateAutoID = True) or when the Update method of the new instance is invoked.

The "LockStyle" argument allows you to indicate the type of lock that will be applied immediately to the new instance. If the WithImmediateAutoID argument is set to False, the LockStyle argument will be ignored.

If the underlying datasource file does not use auto ID generation, the Create method will have the following call signature:

```
Public Shared Function Create(ByVal RepositorySOP As DataRepository, ByVal ID As
String, ByVal LockStyle As LockingStyle) As Product
```

The ID argument allows you to specify the item ID (primary key) of the new instance. If this is not known at create time or needs to be programmatically generated only at Update time, it should be passed in as an empty string. In fact, if this is the case, there is a second overload to the Create method which does not include the "ID" argument.

If you do not supply an item ID at create time, you will need to assign it a value before the Update method is invoked, otherwise an exception will be raised by the Update method.

Singular Class Constructor

An alternative way to create new instances is to used the constructor of the relevant singular class. Each singular class constructor has an identical set of call signatures to that of the "Create" static method of the same class. In fact, if you look at the generated code you will see that the "Create" static method simply invokes the constructor of the same class in order to establish a new instance.

WinForm Data Binding Support

This chapter describes the support for WinForm (rich-client) data binding that is automatically built into each Business Access Class generated using Solution Objects.

Overview

The data binding mechanism supported by .NET WinForm applications allows for data binding to be tied into the data represented by a custom class library. In order for it to do this, the classes contained within such a library must implement various interfaces requirements. The BACs generated by Solution Objects automatically contain such implementations, which means that they may be used freely with all aspects of WinForm data binding.

WebForm Data Binding Support

This chapter describes the support for WebForm data binding that is automatically built into each Business Access Class generated using Solution Objects.

Overview

The data binding mechanism supported by ASP.NET WebForm applications allows for data binding to be tied into the data represented by a custom class library. In order for it to do this, the classes contained within such a library must implement various interface requirements. The BACs generated by Solution Objects automatically contain such implementations, which means that they may be used freely with all aspects of WebForm data binding.

WebForm Data Binding Principles

It is beyond the scope of this guide to give a detailed description of how WebForm data binding works and how a developer may link in and utilize its features. However, in order to explain the support for WebForm data binding that is built into each BAC, below is a brief explanation of the principles of WebForm data binding to custom class libraries.

Within the "Data" group of its Toolbox window, Visual Studio .NET supplies a component called "ObjectDataSource". If you drag and drop this component onto a WebForm, you are able to select a library (and a class within that library) within

your current solution as being a source of data. This data source may then be used to provide data for the WebForm data binding mechanism.

Each ObjectDataSource component uses reflection to inspect the interface of the selected class and allows you to identify the methods that are to be used for data selecting, updating, inserting and deleting.

Once an ObjectDataSource has been created, controls dropped onto the surface of the WebForm can be "bound" to properties of the underlying class using Visual Studio's Properties window.

ObjectDataSource Property Settings

The Visual Studio WebForm ObjectDataSource control has a number of properties that allow you to control its behavior. The web data binding support built into each Solution Objects BAC works best when various ObjectDataSource properties are set as indicated in the table below:

Property Name	Recommended Setting	Description
ConflictDetection	CompareAllValues	This forces optimistic locking to be used when updating existing data items.
EnablePaging	True	Allows the ObjectDataSource to handle the paging of data on the server.
OldValuesParameter FormatString	original_{0}	Identifies the parameter name within the UpdateMethod signature used to data updating. All BAC web update methods have this parameter.

SelectCountMethod	*SelectMethodName*Count	All BAC classes have a method that has the same name as the root selection method with the word "Count" at the end. This Count method returns the total number of items selected and is used by the ObjectDataSource paging mechanism.
SortParameterName	sortedBy	Identifies the parameter name within the SelectMethod signature used to control sorting. All BAC web select methods have this parameter.

The WebDataAssist Control

Solution Objects provides a component that is designed to ease the use of the ObjectDataSource control. It is called the "WebDataAssist" control.

The WebDataAssist control provides the following features which are configurable using its design-time properties:

Property Name	Description
ClientName	Allows the client ID (as displayed in the mv.NET Session Monitor Utility) to be specified.
ConnectionString	Allows the repository connection string to be specified.
ObjectDataSources	The list of ObjectDataSource names to assist. If left blank, all ObjectDataSources on the web page are serviced by this WebDataAssist control.
RetainDataListState	If set to True, maintains the state of the ObjectDataSource's currently selected collection of entities across code-behind execution life-cycles. See below for further details.

The WebDataAssist control also provides the following features which are available as run-time properties/methods

Member Name	Type	Description
DatasourceStateInfo	Property	Returns the data required to reinstate a previously selected collective instance associated with a specified ObjectDataSource.

The WebDataAssist control is located within the following assembly:

```
BlueFinity.mvNET.WebAssist.dll
```

If it is not visible/present within Visual Studio's " mv.NET WebForm " Toolbox group, you may add it using the "Choose Items" right-click context menu option within that Toolbox group.

The RetainDataListState property allows you to request that the WebDataAssist is to automatically persist entity collections (as initially assembled by the ObjectDataSource) across code-behind life-cycles. This has the following benefits:

- Once the original selection list has been assembled, it avoids the need to perform the selection repeatedly each time the code-behind executes. The WebDataAssist control automatically stores the selection list header information in ViewState and uses this to automatically restore/persist the selection list at the start/end of each code-behind execution.

- The persisting of selection lists also includes any data modifications to any collection entries that have not been permanently persisted to the underlying datasource.

- This selection persisting mechanism is compatible with the standard ObjectDataSource paging mechanism.

Web Data Binding Selection Methods

Each static selection method of a BAC is manifested a number of times (as different overloads) on the interface of the singular class. One of these overloads (which is placed within the "Web Data Binding Support" region of the generated code) is decorated with the following attribute:

```
System.ComponentModel.DataObjectMethod(System.ComponentModel.DataObjectMethodType.Select
```

This attribute indicates to the ObjectDataSource control that this method is a candidate for its "SelectMethod" property. Solution Objects ensures that the

signature of this overload is compatible with the requirements of the ObjectDataSource data binding mechanism.

Web Data Binding Maintenance Methods

Each BAC generated by Solution Objects has the following data maintenance static methods on the singular class:

```
Public Overloads Shared Function Update(ByVal ClassName As ClassName) As Int32

Public Overloads Shared Function Update(ByVal ClassName As ClassName, ByVal
original_ClassName As ClassName) As Int32

Public Shared Function Insert(ByVal ClassName As ClassName) As PrimaryKeyDataType

Public Overloads Shared Function Delete(ByVal ClassName As ClassName) As Int32
```

Where *ClassName* represents the relevant BAC singular name and *PrimaryKeyDataType* represents the data type of the item ID property of the class.

Each of the above methods is decorated with the appropriate DataObjectMethod attribute:

```
System.ComponentModel.DataObjectMethod(System.ComponentModel.DataObjectMethodType.Up
date

System.ComponentModel.DataObjectMethod(System.ComponentModel.DataObjectMethodType.In
sert

System.ComponentModel.DataObjectMethod(System.ComponentModel.DataObjectMethodType.De
lete
```

These attributes indicate to the ObjectDataSource control that these methods are candidates for its "UpdateMethod", "InsertMethod" and "DeleteMethod" properties.

Accessing Datasource Data at Run-time

The WebDataAssist control allows you to access the collective instance associated with an ObjectDataSource at run-time by providing the data required to restore the selection from its persisted state.

Below is some example code which shows the data from the currently selected row within a GridView control being accessed to populate 3 TextBox controls on a webform:

	OrgID	Name	OnHold	ZipCode
Select	183	Stationery Logistics Brokers Corp.	☐	2419
Select	684	Pipework Rental Center Corp.	☐	16020
Select	1281	Carpentry Servicing Resellers Inc.	☐	9471
Select	184	Communications Delivery Professionals Inc.	☐	15063
Select	685	Tyre Framework Contractors Corp.	☐	17788
Select	1282	Knitting Design Specialists Inc.	☐	18561
Select	185	Cargo Framework Professionals Corp.	☐	10579
Select	686	Pallet Helpline Center Corp.	☐	19417
Select	1283	Jewellery Design Outlet Corp.	☐	12193
Select	186	Telephone Servicing Services Inc.	☐	6689

1 2 3 4 5 6 7 8 9 10 ...

Name : Stationery Logistics Brokers Corp.

Address : 4283 Hazel Pond
Jesby
SD

Zip code : 2419

Save Cancel

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Private SOPData As BlueFinity.mvNET.SolutionObjects.DataRepository =
                                                                Repository.Initialize()

    Protected Sub GridView1_DataBound(ByVal sender As Object, ByVal e As System.EventArgs)
                                                            Handles GridView1.DataBound

        PopulateControls(RestoreList(0))
        GridView1.SelectedIndex = 0

    End Sub

    Protected Sub GridView1_SelectedIndexChanged(ByVal sender As Object, ByVal e As
                                System.EventArgs) Handles GridView1.SelectedIndexChanged

        PopulateControls(RestoreList(GridView1.SelectedIndex))

    End Sub

    Private Function RestoreList() As Organizations

        Dim Organizations As Organizations
        If WebDataAssist1.DataSource(odsOrganization) Is Nothing Then
            Organizations = Organizations.RestoreFromState(SOPData,
                  WebDataAssist1.DataSourceStateInfo(odsOrganization), GridView1.PageIndex *
                  GridView1.PageSize, GridView1.PageSize)
            WebDataAssist1.DataSourceRestore(odsOrganization, Organizations)
        Else
            Organizations = CType(WebDataAssist1.DataSource(odsOrganization), Organizations)
        End If
        Return Organizations

    End Function
```

```
    Private Sub PopulateControls(ByVal Organization As Organization)

        txtName.Text = Organization.Name
        txtAddress.Text = Organization.AddressText
        txtZip.Text = Organization.ZipCode

    End Sub

    Private Function EditSession() As Organizations

        Dim Organizations As Organizations = RestoreList()
        Organizations(GridView1.SelectedIndex).BeginEdit()
        Return Organizations

    End Function

    Protected Sub txtName_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs)
                                                                    Handles txtName.TextChanged

        EditSession(GridView1.SelectedIndex).Name = txtName.Text

    End Sub

    Protected Sub txtAddress_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs)
                                                                  Handles txtAddress.TextChanged

        EditSession(GridView1.SelectedIndex).AddressText = txtAddress.Text

    End Sub

    Protected Sub txtZip_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs)
                                                                    Handles txtZip.TextChanged

        EditSession(GridView1.SelectedIndex).ZipCode = txtZip.Text

    End Sub

    Protected Sub btnSave_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
                                                                                    btnSave.Click

        RestoreList(GridView1.SelectedIndex).Update()
        GridView1.DataBind()

    End Sub

    Protected Sub btnCancel_Click(ByVal sender As Object, ByVal e As System.EventArgs)
                                                                      Handles btnCancel.Click

        Dim Organizations As Organizations = RestoreList()
        Organizations(GridView1.SelectedIndex).UpdateCancel()
        PopulateControls(Organizations(GridView1.SelectedIndex))

    End Sub

End Class
```

In the above code, "odsOrganization" is the name of the ObjectDataSource control bound to the Organization class; GridView1 is the name of the GridView control which is bound to odsOrganization.

This code assumes that the " RetainDataListState" property of the WebDataAssist control is set to "True". Below is a summary of the purpose/function of each of the code sections above:

Method Name	Description
GridView1_DataBound	This routine handles the DataBound event of the GridView control. It forces the TextBox controls to be populated with data from the first row in the grid.
GridView1_SelectedIndexChanged	This routine handles the SelectedIndexChanged event of the GridView control. It forces the TextBox controls to be populated with data from the currently selected row in the grid.
RestoreList	This routine returns an Organizations instance by either restoring it from the previously persisted state (using the WebDataAssist.DataSourceStateInfo property) or by extracting a previously restored instance (using the WebDataAssist's DataSource function). The restore from state approach passes in the GridView's current page position so that the returned Organizations instance matches that held by the ObjectDataSource.
PopulateControls	Uses data from the currently selected Organization instance to populate the values of the various edit/display controls.
EditSession	Prepares the current Organization instance for editing.
xxx_TextChanged	The TextChanged event handlers for the 3 TextBox controls being used.
btnSave_Click	Saves any amendments to the currently selected Organization instance and forces the GridView data to be refreshed (by using its DataBind method) to display any new values.
btnCancel_Click	Cancels any unsaved amendments to the currently selected Organization instance and forces repopulation of the TextBox controls.

Developing Silverlight Applications

This chapter describes how you can use your business objects layer to create rich-client applications that run inside a Web browser using Microsoft's "Silverlight" product.

Overview

Microsoft's Silverlight product is an environment (plug-in) that runs inside a Web browser. Most of the popular browsers in use today are fully supported. The Silverlight plug-in allows .NET managed code to execute within the sand-boxed browser environment and also supports XAML (WPF) rendering of the user interface. This combination gives us just about everything we need to create applications that have all the look and feel of functionally rich desktop applications but with the added benefit of being able to run as a browser application.

As mentioned above, Silverlight supports (a cut down version of the full) WPF specification and, as such, provides excellent data binding support.

User-based Property Security

This chapter describes the built-in security system provided by Solution Objects. This security mechanism is designed to ease the task of programmatically determining which data fields a user should be shown or allowed access to based on their security rating.

Overview

It is a frequent requirement within applications to control the fields that a particular user is displayed or is allowed to edit within the interface of an application. This requirement is invariably driven either in part or whole by their "security privilege level" - a setting that the application usually needs to maintain as part of its feature set. In order to ease the task of implementing such functionality, Solution Objects provides a mechanism whereby the properties of each entity can be flagged with security information. This information can be accessed by the developer at runtime in order to determine, for example, which controls on a form should be hidden or disabled as necessary.

Security information is held at the entity model version level, this allows security profiling to evolve across different entity model versions. There are 2 main parts to Solution Object's security mechanism:

1) The definition of a list of security groups

2) The association of these security groups to entity properties

Defining Security Groups

The security groups that exist within a particular entity model version are defined within the version details maintenance window, accessed by right-clicking the relevant version node within the Data Manager's treeview area and selecting the "Maintain Version Details" menu option. On the resulting window, select the "Security Groups" tab.

You are then able to add and remove entries from the security group list for this version.

Associating Security Groups with Properties

The association of security groups with properties is done within the Data Access Class maintenance window. Within this window, once you have selected the "Datasource Specific" view, a "Security" tab will be available.

The first thing that you need to define is the style of security control for this entity using the "Class property security style" combobox control. There are 3 options:

- "No security" - no security profiling will be implemented for this entity.
- "Same security for all properties" - you are able to define a single security profile for this entity. This profile will then be applied to all properties within the entity.
- "Custom security per property" - you are able to create an separate security profile for each property within the entity.

Each security profile allows you to specify which of the security groups of the model version are allowed access to a property/properties and, if access is allowed, whether this is full (read/write) access or just read-only.

Using Security Information at Run-time

The creation of security groups and profiles at design-time does not, in itself, cause anything to happen automatically at run-time within an application. It is the responsibility of the developer to:

a) Specify the security group membership of a specific user

b) Implement any user interface/application ramifications of non-membership of a group

However, Solution Objects does provide an easy to use static method in order to determine whether a user has access (and at what level) to a specific property based on their security group membership.

The first thing that you need to do at some point early on in the execution of your application is to construct a string array of security groups that the current user belongs to. How you determine this is down to you to decide.

Each Business Access Class has a static method called "PropertySecurityAccessLevel" which accepts 2 arguments, the first is the string array of security groups that the user belongs to and the second is a (string value) property name. For example, if we have a Business Access Class called "Organization" that has a "Name" property :

```
Private SecurityGroupMembership = New String() {"MANAGER"}

Dim accessLevel As PropertySecurityAccessLevelType =
Organization.PropertySecurityAccessLevel(SecurityGroupMembership,
Organization.MemberNameEquates.Property.Name)
```

Or, in C#:

```
string[] SecurityGroupMembership = new String[] { "MANAGER" };

PropertySecurityAccessLevelType accessLevel =
Organization.PropertySecurityAccessLevel(SecurityGroupMembership,
Organization.MemberNameEquates.PropertyName.Name);
```

From the above you can see that Solution Objects contains an enumeration called "PropertySecurityAccessLevelType". This enumeration can be used to identity the access level that a user has to a specific property, namely:

- None
- ReadOnly
- Full

You will also notice that the above code makes use of the property name equates that are automatically generated for each Business Access Classes, in the above example "Organization.MemberNameEquates.PropertyName.Name".