

# ShelterBase Developer Notes

*ShelterBase* is a database application developed for use by Arizona domestic violence shelters. The 3-tier architecture consists of a database hosted on Microsoft SQL Server, the ShelterBase Server middle tier, and the ShelterBase client. The database and server components are MS Windows service applications installed on the same computer (server), while the client can be installed on the same computer or additional LAN/VPN connected machines.

The client and server components are .NET applications compiled with the 4.6.1 Framework and employing communication via Windows Communication Foundation (WCF). *(The original 2.x versions employed .NET Remoting.)* These components share two assemblies, SBshared.dll and ReportsCommon.dll. SBshared contains the data contracts for the business objects to be serialized as well as the service contract for a shared interface, IDataManager. The reports library is shared to enable references to specific reports which the server uses to retrieve appropriate datasets for those reports.

The application design is optimized for data analysis verses transaction performance. Typical employment is not highly transactional. Communication between the server and database components employs ADO.NET. Connections are opened and closed for each operation, i.e., open connections are not maintained during user interactions and records are not locked. While this can result in conflicts if multiple users attempt to edit the same record, the chances of collision are virtually nil; record timestamps are not used to advise users if a collision takes place during a save operation. Default connection pooling is used to maintain performance for users conducting multiple operations.

The ShelterBase solution is presently developed using Visual Studio 2017 Professional and contains several projects:

* The Client project is a WinForms desktop application used for data entry and reporting.
* The Service project compiles as a Windows service. Clients connect to the service via Windows Communication Foundation for data entry and retrieval. The service connects to the database via ADO.NET to retrieve or manipulate data as requested by the client. *Note: There is also a “Server” project which runs as a Windows application and uses code identical to the Service project. This is used for testing and debugging the code used by the service.*
* The SBshared project compiles as a library assembly and is referenced by both the Client and Service assemblies. It includes one service contract, the IDataManager interface, and several data contracts for objects passed between the Client and Service. It also includes enumerations used by both assemblies.
* The ServerConfigurator project is a simple WinForms application primarily used to create an XML file with startup settings for the Service application. It can also be used to start/stop the service and to create an empty copy of the database.
* In the ReportLibraries folder are [five library assembly projects](#Reports) used for reporting.
* Finally, there are three Visual Studio Intaller Setup projects to install the client, server (Windows service), and configurator applications.

Developer notes are included for:

* [Remaining Issues and Considerations](#Issues)
* [ShelterBase Client](#Client)
* [ShelterBase Reporting](#_ShelterBase_Server)
* [ShelterBase Server](#Server)
* [SQL Server Database](#Database)

## Issues and Considerations

ShelterBase is currently a .NET WinForms application. While there was a plan to migrate to Windows Presentation Foundation after WPF matured, that task was never accomplished due to lack of support for MDI and potential problems retraining users to a new interface.

Separate stored procedures were originally used for inserting and updating records in the database. Many have been updated as "Save" procedures using an @IsNew parameter to determine whether an insert or update is appropriate. However, several types of records still use separate procedures for insert vs. update.

The data contracts all have a boolean property, IsChanged, which is used to flag objects whose properties have unsaved changes. These classes should be revised to implement INotifyPropertyChanged and replace the IsChanged property.

Abuser data is rudimentary. It was originally purposed only to support DES reporting requirements, but that section of the report has long since been discarded. Few shelters actually perform abuser counseling. However, those that provide this service could use better data capture.

SysAdmin users have access to a "Run Script" feature on the Admin menu which allows them to execute SQL scripts. This has been useful for disseminating updates to the back end database, but it represents a security risk. While it has never been misused, it may be advisable to drop this feature.

The ShelterBase database defines different security roles for users, but relies on having users with instance-wide SysAdmin access. While nearly all users only use their SQL Server instance for ShelterBase operations, this arrangement could allow these SysAdmin users to access other databases on that instance. While this limitation is in effect, non-ShelterBase database should be installed on a different SQL Server instance. If this ever becomes a significant issue, a new security role that gives effective admin rights to only ShelterBase objects should be created.

## ShelterBase Client Application

The ShelterBase client, a .NET Windows Forms application, uses a Multiple Document Interface. The MDI parent window is used to host both single and multiple instance windows. It connects to ShelterBase Server via TPC over LAN/VPN, or shared memory when installed on the same computer as the server. Two static classes are used by the ShelterBase client to control window properties.

### Settings Class

The Settings class uses two XML files to persist application properties. The information to connect to the server, i.e., server computer name, TCP or shared memory, port number, and user authentication type, is saved in an XML file in isolated storage. This information is the same for all Windows accounts on the client computer. If the file is not detected on application startup the user is advised and allowed to create the connection settings. These settings are saved to the XML file only after a successful connection has been established.

The Settings class also persists the location and size (for sizable forms) of child windows within the MDI parent window. The class also recalls information about which reports from the report libraries are used to populate a reports list for each user. Finally, if the user has entered settings for automated backup of the database to a local, encrypted/compressed file, those settings are also saved and recalled by the Settings class.

### FormManager Class

The FormManager class is used to control instances of multi-instance windows: Abuser, Participant, Child, Community Member, and Intake records. If the window for the specified record is already open it is activated; otherwise, a new window instance for the record is opened. Additionally, as these windows contain lists of related records, the FormManager class is called to retrieve updated lists when the associated records have been updated in another window. *Note: This function was previously performed using events; it became much simpler to manage the windows with a dedicated static class.*

### Custom Controls

The *DatePicker* control, which inherits from DateTimePicker, is used for date entry and supports null values. It adds a property, NullText, which it displays when the date value is null. Otherwise, it displays the entered date value in short date format. The control only returns a date from entries, i.e., the datetime value only has the date component.

The *tabChildIntake* control inherits from TabPage. It contains all the data entry controls for capturing child intake data. A tabChildIntake control is added at runtime to the tab collection of the parent's intake window for each child.

### Operation

Several operations are performed at application start.

If the installation is from a ClickOnce deployment, the update site is checked for a new version.

The Settings class is used to obtain server connection information (if none exists, the user is offered the opportunity to create it). The connection data is then used to create a channel for the global DataManager object, which inherits from IDataManager, and the connection is tested.

* The database revision number is tested for currency and the user is advised if an update script should be applied.
* The user's security role is retrieved and persisted.
* A "Lookup" dictionary of type Dictionary, i.e., a dictionary of dictionaries, is retrieved as a global variable. The included dictionaries are used to populate lists in different windows with entries from most of the data integrity tables. This is accomplished at startup to limit repeated retrieval of the same values during a client session.

After the above items are completed the application is launched by opening the parent MDI window, which configures menus and the toolbar based on the user's security role. Also, the window used for finding clients is opened but kept hidden. When loaded the find client window asynchronously fills four lists with current Participants, Children, Abusers, and Community Members. *Note: "Current" clients include all clients whose intakes were active in the previous 30 days, and Community Members whose records still indicate current.*

### ShelterBase Reports

Five library assemblies are used to generate most of the ShelterBase reports.

**ReportsCommon**, which is a shared assembly for the client and server components, includes a public enumeration of reports. The server uses the enumerations to select the appropriate stored procedure to retrieve a dataset for the indicated report. The assembly also has a public method that returns a datatable of available reports and their descriptions. The ShelterBase client uses this method to populate a list of available reports in the reports window.

**DesReports**, **FunderReports**, and **ShelterReports** expose methods to return report description files (rdlc) as a MemoryStream, as well as methods for generating ReportDataSource arrays for the selected reports. The assemblies essentially operate identically and were originally combined. However, as report requirements increased their functionality was separated to reduce individual dll file sizes for upload/download considerations.

**ReportManager** has methods to select which of the above assemblies are used to supply report definitions and data sources. It also includes the report definitions and datasource methods to produce the “Generic Report” and Survey Results report.

Reports are added to the program with the following steps:

1. A report is first added to the ReportsCommon.Reports enumeration.
2. A datarow with the report’s enumeration, name, default timeframe, and description are added to the ReportsCommon.AvailableReports method.
3. If filters are to be applicable to the report, they are added to the ReportsCommon.ReportFilters list.
4. The report description (rdlc) and dataset (xsd) are added to the appropriate library assembly (DesReports, FunderReports, or ShelterReports).
5. In the rdlc report designer, add the dataset to the Report Data Datasets. Also, add four parameters: StartDate (datetime), EndDate (datetime), OrgName (text), and AppliedFilters (text). Allow blank values for the text parameters.
6. Add the report enumeration to the appropriate library’s RdlcFile method.
7. Create a method with a return type of ReportDataSource[] in the appropriate library, and add that method to the library’s GetReportDatasource method.
8. Add the report enumeration to the ReportManager’s GetRdlc and GetReportDatasource methods.

The libraries’ GetReportDatasource methods can be supplied with datasets from the database in two ways. The preferred method is to add a stored procedure, prefixed by “rpt\_”, that has parameters of StartDate, EndDate, and ReportNumber. The IDataManager interface has a method of GetReportData with a return type of Dataset, so the report must be added to that method in DataManager class of the server. Note that this requires compiling and reinstalling the server component as well as running an update script on the database to add the stored procedure.

Alternatively, IDataManager has two methods that accept SQL strings, GetDataTable and GetDataSet which return a datatable or dataset, respectively. These methods can be used to expediently add a report without modifying the server or database. Since the client only supplies dates and a report enumeration to the affected methods, it is relatively safe from SQL injection. However, best practice is to convert such reports to the stored procedure methodology during the upgrade cycle.

In many cases the report libraries’ ReportDataSource methods manipulate datatables after they have been retrieved from the database. This has the benefit of sharing the processing overhead between the client and the server, especially when compiling data for some of the funder reports.

### ShelterBase Server

ShelterBase Server is compiled as a Windows service with a startup type of automatic. It contains the following classes:

**Program:**  the entry point for the application which starts the ShelterBaseServer as a service.

**ShelterBaseServer:**  inherits from ServiceBase; used to support the Windows service OnStart and OnStop methods. The OnStart method's primary task is to establish the WCF endpoints specified in the settings file created by the ShelterBase Server Configuration application and open the DataManager service host.

**DataManager:**  inherits from the IDataManger interface defined in the SBshared library. In addition to the interface, the SBshared library also defines the contracts for the data objects shared between the server and client applications.

**Lookups:**  a static class that provides methods to create a number of dictionary objects of type <string, int>. These represent the contents of the first two columns of several data integrity tables, which are ultimately passed back to the client as a single dictionary of type <string, dictionary<string, int>>. This technique is employed to limit trips to the database for key/value combinations that rarely change. Instead the entire set of dictionaries are passed to the client on startup. The lookup dictionaries are populated when the a user logs in and when a dictionary entry has been added or edited.

**Globals:**  a static class used to persist server properties supplied during client log ins.

The public methods of the DataManager class, which implement the IDataManager interface, belong to these general categories:

* *Admin methods* for operations such as log in, database backup/restore, and user management.
* *Report methods* used to retrieve datasets supporting client-side reports.
* *Data Retieval methods* that return data objects, e.g., Participants, or datasets used to populate client-side lists. Some methods return tuples of more than one type of data object. Methods that perform data retrieval from the database all use a DataReader object; SqlDataAdapter is not used. The DataReader is then supplied to a DataSet or DataTable Load method, or is read one record at a time to set the properties of a data object to be returned by the method.
* *Save methods* used to insert or update database records. These usually return a struct (defined in SBshared) of type "SaveResult", which includes a boolean indicating successful save, a string to return an error message, a datetime with the insert/update date, and an int value for the generated primary key of the saved record. The save methods usually include a parameter of type SaveArg, a struct defined in SBshared that includes a string for the connection string and a string for the user name, which is stored in a column of the affected record(s).
* *Delete/Merge/Restore/Eradicate methods* perform the following:
  + Mark records as deleted by completing DeletedBy and DeletedDate columns for the indicated record.
  + Combine records from data integrity tables and updating all related records containing the eliminated foreign key.
  + Restore records that have been flagged as deleted. *Admin users only.*
  + Permananetly delete records from the database. *Admin users only.*
* *Private utility methods* for internally manipulating data.

*Note: Many of the public data manipulation methods have a return type of MethodResult, a struct defined in SBshared with a boolean for success and a string for an error message.*

### Database Design

The ShelterBase database, hosted on Microsoft SQL Server, is composed of numerous primary tables, intersection tables, and data integrity tables. The SQL Server instance is configured for "Mixed Mode" authentication.

Data integrity tables containing a primary key integer column and a varchar description are usually named with an "x" prefix. They consist of three types:

* Fixed entries that are unlikely to change, conforming to funder reporting requirements and the generalized business model. While these tables can be joined to other tables, their primary key is NOT used as a foreign key. Instead, potentially "related" tables have check constraints limiting values to those in the integrity tables, which include: xCallType, xCitizenship, xClientType, xCounselingResult, xCounselingStatus, xCounty, xCourtOutcome, xEmployment, xHousingType, xIntakeType, xMaritalStatus, xRequestOutcome, xReservation, xReturnStatus, xShelterType, and xSubstanceProblem.
* Tables with fixed entries conforming to reporting requirements that allow user-entered "Other" values. The required entries have a key value less than 100, while user-created entries have a key value greater than 100. These include: xAbuserRelationship, xCallSubject, xEthnicity, xExitDestination, xExitHousing, xExitReason, xHotlineReferral, xReasonDenied, xReferralSource, xReferralType, and xPriorResidence.
* Lookup tables containing only user-entered values. These are employed to permit reuse of the same selection for: xGoodType, xLanguage, xLawEnforcementAgency, xLocation, xPetType, xReligion, xServiceLocation, xTask, xTribe, xTopic, and xTripPurpose.

*Note: Many of the integrity tables correspond to obsolete reporting requirements but have been retained by user request and for legacy purposes.*

### Database Security Roles

The beta version of ShelterBase 2.x allowed for both Windows authentication as well as mixed mode. During testing administrators of domain networks had difficulty managing access rights, so the application was converted to use only mixed mode. On domain networks the option still exists to use domain controller authentication while SQL Server effects authorization.

ShelterBase users can be assigned to one of five security roles to limit their access to data. Their permissions to tables and stored procedures are explicit (except for SysAdmin users).

* *SysAdmin* users have full access to all ShelterBase objects. This is actually a SQL Server role, not a database role, so these users also have full access to all objects in the SQL Server instance.
* Users in the *ClientDataUser* role can view/edit all client information, but can only enter/edit basic staff information.
* A *StaffDataUser* can view/edit all client data as well as potentially confidential staff information.
* A *ReadOnlyUser* can view all information, but cannot enter/edit records.
* A *LimitedUser* can enter services, referrals, bed assignments, and basic intake records. Access to extended intake information is not allowed.
* A *HotlineUser* can only enter hotline call information.

### Tables

A [complete listing](Tables.html) of the database tables includes the table names, columns and data type, primary keys, indexes, foreign keys, and other constraints.

### Stored Procedures

Nearly all operations performed on the database are accomplished with [stored procedures](Procedures.html). Typical CRUD procedures are prefixed with "proc\_", while those used to retrieve data for reports have an "rpt\_" prefix.

### Schema

ShelterBase table relationships, excluding data integrity tables, are depicted with most relevant columns. Primary keys are indicated in boldface; foreign keys to data integrity tables have a "-" prefix. Intersection tables have a dashed border. Enforced relationships are indicated with solid lines, while relationships that depend on a value in addition to the key value are shown with dashed lines.

[Participants, Children, Intakes, Shelters](#Participants)

[Hotline Calls](#Hotline)

[Services](#Services)

[Referrals](#Referrals)

[Abusers](#Abusers)

[Surveys, Outcomes, Self-Sufficiency Matrix](#Outcomes)

[Staff](#Staff)

[Grants and Grantors](#Grants)

[Pets and Information Release Permissions](#Pets)

