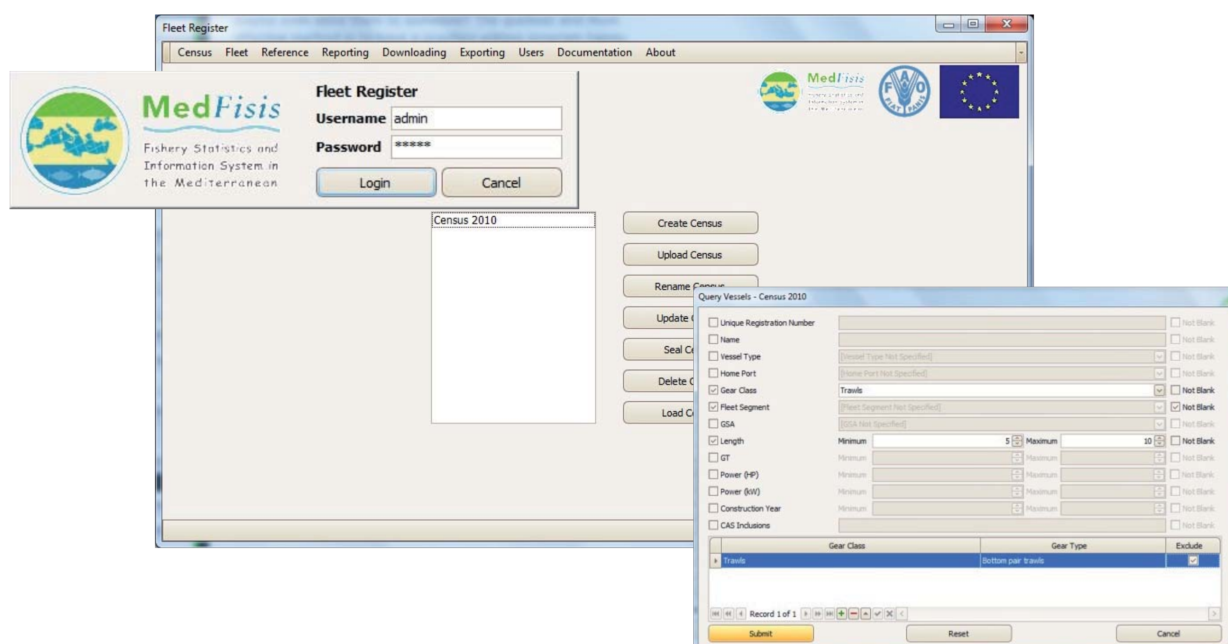
MedFisis
Fishery Statistics and Information System in the Mediterranean

MedStat 2011 – Fishing Vessel Census

FISHING VESSEL REGISTER SOFTWARE TECHNICAL DOCUMENTATION



By

Dawn Borg Costanzi

MedFisis technical document

FAO, Rome 2011

Fishery Statistics and Information System in the Mediterranean
MedFisis Project (Year 3) GCP/INT/918/

The designations employed and the presentation of material in this information product do not imply the expression of any opinion whatsoever on the part of the Food and Agriculture Organization of the United Nations (FAO) concerning the legal or development status of any country, territory, city or area or of its authorities, or concerning the delimitation of its frontiers or boundaries. The mention of specific companies or products of manufacturers, whether or not these have been patented, does not imply that these have been endorsed or recommended by FAO in preference to others of a similar nature that are not mentioned.

The views expressed in this information product are those of the author(s) and do not necessarily reflect the views of FAO.

ISBN 978-92-5-107144-1

All rights reserved. FAO encourages reproduction and dissemination of material in this information product. Non-commercial uses will be authorized free of charge, upon request. Reproduction for resale or other commercial purposes, including educational purposes, may incur fees. Applications for permission to reproduce or disseminate FAO copyright materials, and all queries concerning rights and licences, should be addressed by e-mail to copyright@fao.org or to the Chief, Publishing Policy and Support Branch, Office of Knowledge Exchange, Research and Extension, FAO, Viale delle Terme di Caracalla, 00153 Rome, Italy.

© FAO 2012

Abstract

The FAO programme MedStat, developed within the activities of the MedFisis project, provides a methodical approach to the challenge being faced by a country which is planning to undertake a National Fishing Fleet Census. The present document is part of a series of documents providing guidance on the implementation of a Fishing Fleet Census, as the technical documentation which goes hand in hand with the MedStat Fishing Vessel Register Software. It provides guidance to developers and technical experts who would like to understand the way in which the software package is designed and built and how it should be managed. Many practical and procedural details are included, in order to give a wide knowledge base to those who intend to customise the program for use in a particular country or situation, or those who would like to develop extensions or enhancements for the application.

For bibliographic purposes this document should be cited as follows:

Borg Costanzi D. 2011. MedStat 2011 – Fishing Vessel Census; Fishing Vessel Register Software: Technical Documentation. GCP/INT/918/EC/MedFisis - MedFisis Technical Document, 67 pp

Acknowledgements

I would like to show my warmest appreciation to Mr. Salvatore Coppola, for his vision and management for the software package, and also Ms. Alicia Mosteiro Cabanelas, for both of whose guidance and support I am truly grateful. Without these MedFisis members, the project would not have been successful and the development, and especially debugging and improvement, of the Fleet Register application would not have been possible.

Much gratitude goes to Dr. Matthew Camilleri from the General Fisheries Commission for the Mediterranean. His constant backing, sound advice and practical experience were fundamental to my work, and also to the success of the project as a whole.

I also give sincere thanks to Mr. Roberto Emma, also from the General Fisheries Commission for the Mediterranean, for his invaluable technical guidance, all the hard work he also put into the development of the application and his drive to see a complete and perfected software package in place.

Special thanks to Dr. Derek Pilgrim for his patience throughout testing and valuable contribution from his professional experience.

Contents

Abstract i
Acknowledgements ii
Contents iii
Figures v
Tables vi
Technical Overview 1
Application Structure 1
Application Modules 2
Flow of Information 3
Preparation for Client-Server system 10
Database Structure 11
Register Tables 11
Reference Tables 24
Fleet Register Configurator 31
Country-related Settings 31
SQL Server Parameters 31
Reference Table Configuration 32
Labels Configuration 32
Existing Database and Language Translations 32
Developer's How-To Guide 34
Adding Fields 34
1. Update the Database 34
2. Extend the Data Access Layer 35
3. Modify the Communications Classes 36
4. Revise the User Interface Forms 36
5. Extend the Validator Utility 39
Customising Reference Tables 40
Setting National Tables 40
Migrating Data 40
Customising the Visual Aspect 41
Modifying Integrity Checks 42
1. Insert the Data into the Database 42

2. Update the Data Access Layer 43
3. Extend the Communications Class 43
4. Add to the User Interface 44
5. Extend the Validator Utility 44

Modifying Consistency Checks 45

Adding Reports 46

1. Ensure Database View covers Requirements 46
2. Create a Results Table in the Data Access Layer 46
3. Extend the Communication Class 47
4. Draw Up the Report 47
5. Provide Access to the Report through the User Interface 48

Adding Export Routines 49

1. Optionally Create a Database View 49
2. Insert a New Data Set 49
3. Extend the Communication Class 49
4. Generate the Export File within the User Interface Layer 50

Adding a Language 51

1. Insert the Translation in the XML file 51
2. Update the Language Data Set 51
3. Set the Fleet Register Configurator 51

General Overview of Localisation System 52

Adding a Setting to the Configurator 53

1. Add the Value to the Settings File 53
2. Update the Configurator Interface 53
3. Save the Setting 53

Creating an Installation Package 54

1. Primary Outputs 54
2. Content Files 54
3. Custom Actions 55
4. Prerequisites 55
5. Setup Properties 55

Further Comments or Queries 56

References 57

Figures

Figure 1 – Fleet Register application structure 1
Figure 2 – Register ERD 12
Figure 3 – Vessel ERD 13

Tables

Table 1 – Users table 14
Table 2 – Census table 14
Table 3 – Vessel table 14
Table 4 – Vessel Recording table 15
Table 5 – Vessel Characteristics table 16
Table 6 – Vessel Structural Characteristics table 16
Table 7 – Vessel Authorisation table 17
Table 8 – Vessel Gear table 17
Table 9 – Vessel Engine table 18
Table 10 – Vessel Electronics table 18
Table 11 – Vessel Electronic Equipment table 18
Table 12 – Vessel Deck Machinery table 18
Table 13 – Vessel Winch table 19
Table 14 – Vessel Ownership table 19
Table 15 – Vessel Crew table 20
Table 16 – Vessel Port table 20
Table 17 – Vessel Operation table 20
Table 18 – Vessel Pollution Prevention table 21
Table 19 – Vessel Remark table 21
Table 20 – Owners table 21
Table 21 – Vessel Modification Types table 22
Table 22 – Vessel Modifications table 22
Table 23 – Vessel Deletions table 22
Table 24 – Vessel Unique Registration Number table 22
Table 25 – Check Range table 23
Table 26 – Check Quality table 23
Table 27 – Check Failure table 23
Table 28 – Basic reference table 24
Table 29 – List of basic reference tables 25
Table 30 – List of basic reference tables currently not in use 25
Table 31 – Countries reference table 26
Table 32 – Crew Coops reference table 26
Table 33 – Equipment reference table 26
Table 34 – Events reference table 26
Table 35 – EU Events reference table 26
Table 36 – FAO Statisticl Divisions reference table 27
Table 37 – Fleet Segmentation reference table 27
Table 38 – Gears reference table 27
Table 39 – Gears reference table 27
Table 40 – Geographical Sub Areas reference table 27
Table 41 – Group of Species reference table 27
Table 42 – Hull Materials reference table 27

Table 43 – LOA Classes reference table 28
Table 44 – Minor Strata reference table 28
Table 45 – Ports reference table 28
Table 46 – Species reference table 28
Table 47 – Substrata reference table 28
Table 48 – Vessel Types reference table 28
Table 49 – Vessel Types National reference table 29
Table 50 – Fleet Segment Identification reference table 30
Table 51 – Gear-Group of Species reference table 30
Table 52 – T-Distribution reference table 30
Table 53 – Vessel-Gear Types reference table 30

Technical Overview

This section is intended as a general idea of the way in which the Fleet Register is built and the way in which it functions. Details are not given of each part of the application, but a few typical examples are taken to give a good overview of the various aspects to be understood by developer's intending to modify or extend the system.

Application Structure

The MedFisis Fleet Register is an application built for countries in the Mediterranean to be able to manage statistical information regarding the vessels within the national fleet and their characteristics, export data and produce the required national and regional reports.

The application is built upon the .NET framework (version 3.5 SP1) in the C#.NET programming language, using Visual Studio .NET 2008. The Developer Express (DevExpress v2009 vol 3) tools were also used for enhanced user interface components and to facilitate development.

The following diagram gives an overview of the multi-tier structure of the Fleet Register application:

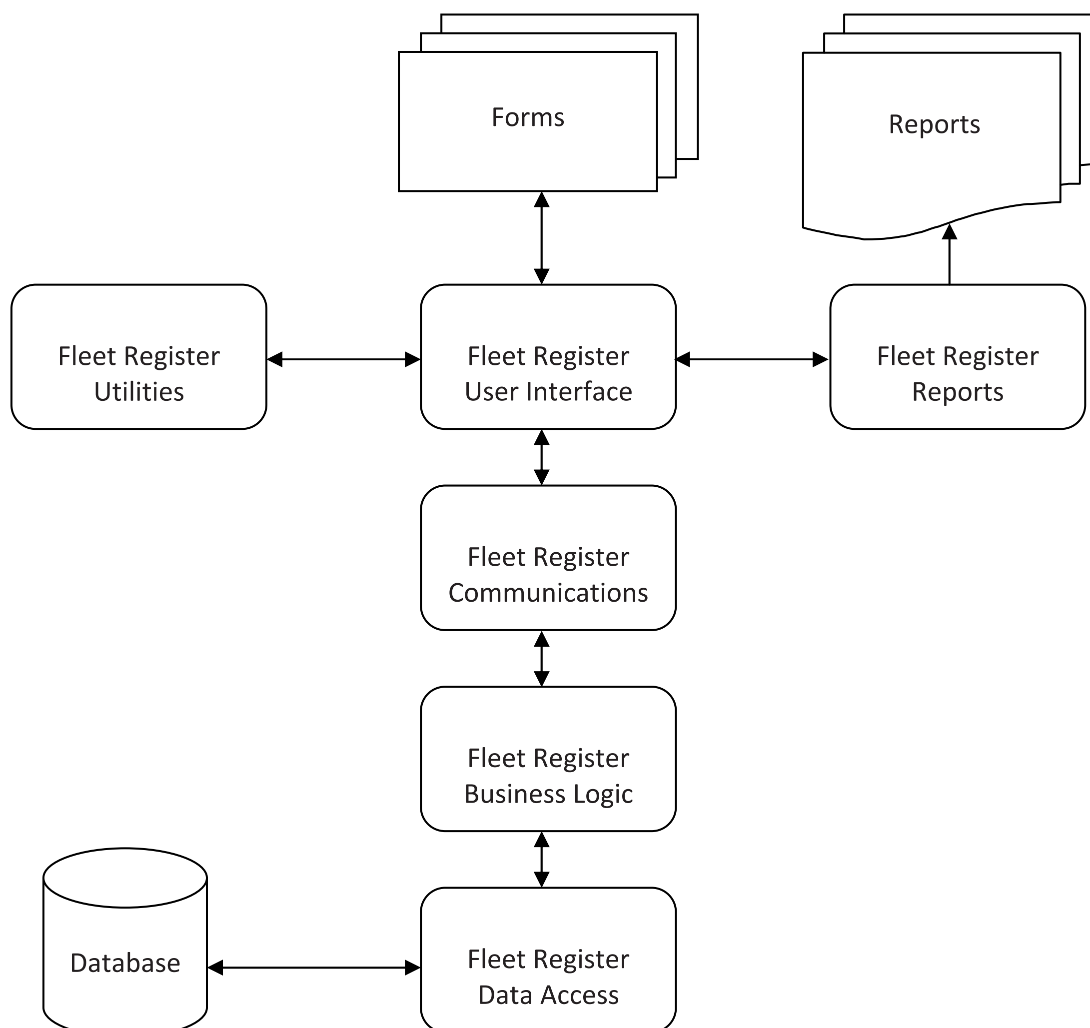


Figure 1 – Fleet Register application structure

Application Modules

Database

The entire application is based on a single database, namely the *FleetRegister* database, which is described in detail in the following section.

Fleet Register Data Access

The data access layer, namely the *FleetRegisterDA* project, is a collection of data sets which interact directly with the tables and views within the database. These data sets generally follow the table structure within the database, although some join data from various tables into a single collective structure, for direct use by the objects within the application.

Fleet Register Business Logic

The business logic layer, namely *FleetRegisterBL*, contains the system-generated objects which precisely reflect the data access layer. The various data tables and data rows reproduce the schemas and relationships modelled by the data seta in the data access layer, and are used by the various components of the application to package and transport the data between different levels.

Fleet Register Communications

The Communications layer, specifically the *FleetRegisterComm* component, is that which links the user interface layer to the data access layer, using the business logic objects. It provides all the various methods required to select from, update, insert to and delete from the data objects, with minor additional logic to ensure that the information is sent by and returned to the user interface component in the simplest way possible to avoid excessive processing.

Fleet Register User Interface

The *FleetRegisterUI* module is the most important of the system, in that it includes the majority of the functionality. It is concerned with the display of information to the user and the acceptance and processing of user commands in order to carry out particular tasks, through the various forms, built using the DevExpress *XtraForm* components. It interacts with the Utilities module in order to perform commonly-repeated actions and with the Reports module to generate reports. The main component within the user interface layer, it invokes the database routines through the use of the communications methods.

Fleet Register Utilities

This component includes a number of functions which are commonly used within the user interface layer, particularly logging, so that the user will have a record of show-stopping issues on the local file system, validation methods and the display and closure of the wait form.

Fleet Register Reports

This module deals completely with the display and layout of the various reports generated at the user interface layer, which are all built using DevExpress *XtraReports* tools and components.

Flow of Information

The flow of information across the various layers of the application is demonstrated by a number of examples taken directly from the application. These code samples will illustrate the way in which the application handles the various database functions (insert, update, delete and select), highlight the tasks handled by each different module and also give some insight into the programming techniques used throughout.

Insertion of Information

The insertion of information will be shown by taking the example of adding a new vessel owner or company to the database.

The insertion is initiated at the user interface level, through the main form *frmMain*, where the user can click the *Reference* > *Companies* > *New Company* menu item or the *New Company* button, once the Companies are loaded. Another option is through another form, *frmVesselDetails*, by loading the census, loading the fleet and then modifying a particular vessel, and clicking the *New Owner* button on the *Ownership* tab. Either of these user actions triggers the instantiation of *frmOwnerDetails*, which is then displayed for the owner to insert all the relevent details. At this point, after validation, all the required information is available at the user interface layer within the *frmOwnerDetails* form.

This same form contains a method, *GetOwnersTable()*, which deals with the data to be transferred. This method is called from *frmMain* or *frmVesselDetails*, to which an instance of the *FleetRegisterBL. VesselDataSet.OwnersDataTable* class, an object from the business logic layer, is returned. The *GetOwnersTable()* method first initialises a *FleetRegisterBL.VesselDataSet.OwnersRow* object, packages all the owner or company information within it, also setting default values for the data which is not available. The row is then added to new vessel data set's *OwnersDataTable* object, a copy of which is returned out of the method. Part of the code is shown below for full comprehension.

```
public FleetRegisterBL.VesselDataSet.OwnersDataTable GetOwnersTable()
{
   FleetRegisterBL.VesselDataSet dsVessel = new FleetRegisterBL.VesselDataSet();
   FleetRegisterBL.VesselDataSet.OwnersRow owner = dsVessel.Owners.NewOwnersRow();
   if (originalOwner != null)
      owner.OwnerID = originalOwner.OwnerID; //this is for update
   else owner.OwnerID = -1; //this is for insert
   if (lkpCompanyType.EditValue == null)
      owner.CompanyType = "";
   else owner.CompanyType = lkpCompanyType.EditValue.ToString();
   owner.CompanyName = txtCompanyName.Text;
   if (spnEstablishedYear.EditValue == null)
      owner.EstablishedYear = -1;
   else owner.EstablishedYear = int.Parse(spnEstablishedYear.Value.ToString());

   ...

   owner.Remarks = txtRemarks.Text;
   dsVessel.Owners.Rows.InsertAt(owner, 0);
   return (FleetRegisterBL.VesselDataSet.OwnersDataTable)dsVessel.Owners.Copy();
}
```

Back in the main form or the vessel details form, this *OwnersDataTable* object is immediately passed to the *InsertOwner()* method of the *OwnerCommunications* class within the *FleetRegisterComms*. *OwnerCommunications* is constructed as a static class, which exposes the *Instance* property to give access to the private and static *OwnerCommunications instance*. Therefore, the call to the *InsertOwner()* method is as follows:

```
int ownerID = OwnerCommunications.Instance.InsertOwner(owners);
```

This method, shown below, simply unpackages all the data from the business logic object, and uses the vessel's table adapter manager (which is generated automatically) to make a call to the *OwnersTableAdapter* (which was instantiated within the *OwnerCommunications* constructor) to insert the owner at the data access layer.

```
public int InsertOwner(FleetRegisterBL.VesselDataSet.OwnersDataTable table)
{
   try
   {
      FleetRegisterBL.VesselDataSet.OwnersRow row =
         (FleetRegisterBL.VesselDataSet.OwnersRow)table.Rows[0];
      return int.Parse(tamngVessel.OwnersTableAdapter.InsertOwner(
         row.CompanyType, row.CompanyName, row.EstablishedYear, row.Address,
         row.Postcode, row.Town, row.Region, row.Country, row.Telephone,
         row.RegistrationOffice, row.Cooperative, row.CooperativeName,
         row.Syndicate, row.SyndicateName, row.Remarks).ToString());
   }
   catch
   {
      return -1;
   }
}
```

At the data access layer, the *VesselDataSet* includes an *Owners* data table, which is an exact representation of this same table within the database, and an *OwnersTableAdapter*, which provides all the queries to access that particular table. One of these queries is the *InsertOwner* query, whose SQL command is shown below, and which accepts all the owner data as parameters and inserts a new record into the table, returning the newly-created owner's identification.

```
INSERT INTO [dbo].[Owners]
([CompanyType], [CompanyName], [EstablishedYear], [Address], [Postcode], [Town],
[Region], [Country], [Telephone], [RegistrationOffice], [Cooperative],
[CooperativeName], [Syndicate], [SyndicateName], [Remarks])
VALUES (@CompanyType, @CompanyName, @EstablishedYear, @Address, @Postcode, @Town,
@Region, @Country, @Telephone, @RegistrationOffice, @Cooperative, @CooperativeName,
@Syndicate, @SyndicateName, @Remarks);
SELECT SCOPE_IDENTITY()
```

Within the *InsertOwner()* method of the communications layer, the returned ID is converted to an integer, or the value of -1 in the case that no valid ID is returned. This value is then received within the user interface class and used to determine whether the insertion was successful or not, which messages are to be shown to the user, and which further action is to be taken.

Updating of Information

The updating of information within the database will be illustrated using the example of changing an application user's password.

All the user's details, including the password, are stored within the *FleetRegister* database, in a table called *User*. Therefore, within the data access layer, in *FleetRegisterDA*, there is a *User* data set, which reflects this same structure, and a *UserTableAdapter* which provides access to the table. One of the SQL queries provided is *UpdatePassword*, which accepts a user ID, which is the record identifier, and a password string as parameters, and updates the password of that particular user as follows:

```
UPDATE [dbo].[User] SET [Password] = @Password WHERE [UserID] = @UserID
```

The *UserCommunications* class within the *FleetRegisterComm*, which follows the same general structure as *OwnerCommunications* class in the previous example, contains the matching *UpdatePassword()* method, which calls the above query as follows:

```
public bool UpdatePassword(int userID, string password)
{
   try
   {
      tamngUser.UserTableAdapter.UpdatePassword(password, userID);
      return true;
   }
   catch
   {
      return false;
   }
}
```

In this case, since no complex data structures are necessary, no objects from the *FleetRegisterBL* class are created or used during the interaction between the communications and data access layers. The very simple method is designed so that an error in performing the update will result in a false value being returned, to indicate a failure in updating the password. Therefore, at the user interface level, this determines which message to show the user, as shown below:

```
bool updated = UserCommunications.Instance.UpdatePassword
   (currentUserID, dataForm.Password);
if (updated)
   XtraMessageBox.Show(languageCommunications.GetString
      ("strPasswordChangedSuccessfully"));
else XtraMessageBox.Show(languageCommunications.GetString
      ("strstrPasswordNotChangedPleaseTryAgain"));
```

This code snippet forms part of the method invoked when the user clicks on the *Users > Change Password* menu item, which automatically refers to the user currently logged in, whose ID is stored globally at the time of successful login. The new password is gathered by creating and showing the user an instance of the *frmUserDataEntry* form, named *dataform*, which would have been customised upon loading for password modification.

Therefore, in this case as in general, the user interface layer is both the beginning, through the menu item click, as well as the end point, by showing the appropriate user messages, for the user task at hand.

Deletion of Information

When speaking about deletion within the Fleet Register, one must consider that there are two types of deletion:

- an actual and permanent deletion of data, which is usually system data and invoked by the system itself
- a more temporary deletion of user data, in a way which allows the user to see which data has been deleted and possibly allow it to be restored.

The way in which each form of deletion takes place will be shown below using two different examples, in order to demonstrate also the differences between them.

The first example which will be taken is that of permanent deletion of saved data related to failures of data integrity checks.

These check failures would have been saved to the system when a particular vessel would have been inserted and modified and not found to satisfy all the range and quality checks, or when a full census data integrity scan would have taken place, and the results of all failures of the census saved to the database. On the other hand, the entire set of check failures for a census are deleted before a data integrity scan takes place, in order to start off with a clean slate, and therefore this function is not directly invoked by the user.

The *Data Integrity Check* menu item is that clicked by the user to start the data integrity scan, if enabled through the Fleet Register Configurator. The user interface event invoked immediately makes a call to the communications level *CheckCommunications* class, as follows. The parameter is the identifier of the census whose failure data is to be deleted, which is that currently loaded.

```
CheckCommunications.Instance.DeleteFailuresByCensusID(currentCensusID);
```

Within the communications class, the method is a simple one which iterates through all the failures for that census and deletes them. Two objects from the business logic level are utilised: *CheckFailureDataTable* and *CheckFailureRow* as contained within it. Two database queries from the data access *CheckDataSet* class are also used, both of which use the *CheckFailureTableAdapter* to modify the *CheckFailure* table in the database: *GetFailuresByCensusID,* indirectly through the *SelectFailuresByCensusID()* method and *DeleteFailureByFailureID*. This method is show below for clarity's sake.

```
public bool DeleteFailuresByCensusID(int censusID)
{
   try
   {
      FleetRegisterBL.CheckDataSet.CheckFailureDataTable dtFailures =
         SelectFailuresByCensusID(censusID);
      foreach (FleetRegisterBL.CheckDataSet.CheckFailureRow row in dtFailures.Rows)
         tamngCheck.CheckFailureTableAdapter.DeleteFailureByFailureID
            (row.CheckFailureID);
      return true;
   }
   catch
   {
      return false;
   }
}
```

In the same way as in previous example, the communications class returns a value to the user interface level to indicate success or failure of the action invoked, in order to be able to proceed.

The second example that will be taken, to illustrate the way in which data may be expired instead of deleted, will be that of removing a census.

The user may initiate the removal of a census from the census list visible upon load of the application by selecting a census within the list and clicking the *Delete Census* button or menu item. This user interface trigger, after being confirmed by the user, invokes the *ExpireCensus()* method in the *CensusCommunications* class, passing the ID of the census selected as a parameter and the ID of the current user logged in, to be able to recall who removed the census:

```
bool expired = CensusCommunications.Instance.ExpireCensus
   (int.Parse(lstCensus.SelectedValue.ToString()), currentUserID);
```

At the communications level, the method called is a straightforward direction to the data access layer, specifically the *CensusTableAdapter* of the *CensusDataSet*, as shown below:

```
public bool ExpireCensus(int censusID, int userID)
{
   try
   {
      tamngCensus.CensusTableAdapter.ExpireCensus(DateTime.Now, userID, censusID);
      return true;
   }
   catch
   {
      return false;
   }
}
```

The *ExpireCensus* query used is nothing more than an update query, in which the census identified by the last parameter has its *Expired* value set to today's date and the *ExpiredUserID* set to the user currently logged in. A successful expiration results in the user being alerted by a message in the user interface and a reloading of the census list, so that the newly-expired census, like all other previously removed ones, will not be visible to the user.

Expiring a data record and retaining all the information in an invisible manner allows for the potential listing of all removed items, by finding those whose *Expired* value is in the past, knowing when the items was deleted and by who, and leaves the possibility of undoing the expiration, by resetting the *Expired* value, and ideally also the *ExpiredUserID*, when present.

Selection of Information

The querying of the database for a particular set of vessels based on user criteria will be taken in order to detail the flow of information in selection of information.

The vessel information is split up into a number of vessel tables within the database, as shown in the Register Tables section, which follows on page 13. The database also contains a *QueryView* view, which represents the all of the vessel information from the various tables in a flat structure which allows for easier querying of information.

The data access level contains a *VesselSummary* data set, which in turn defines the *QuerySummary* table, which is that used to store the selected results. This table, unlike those seen in the previous examples, does not directly reflect a database table structure, but it comprises the fields deemed to be most important to identify a particular vessel. The *QuerySummaryTableAdapter* linked to the table contains the basic query used to retrieve the data, in two different versions: one for English language and one for national language of the reference data, which is shown here:

```
SELECT
[VesselID], MAX([UniqueRegistrationNumber]) AS [UniqueRegistrationNumber],
MAX([FisheryDepartmentRegistrationNumber]) AS
[FisheryDepartmentRegistrationNumber],
MAX([RegistrationNumber]) AS [RegistrationNumber],MAX([PortName]) AS [HomePort],
MAX([Name]) AS [VesselName], MAX([OperationalStatusName]) AS [OperationalStatus],
MAX([VesselType]) AS [VesselType], MAX([GT]) as [GT], MAX([Length]) as [Length],
MAX([EnginePower]) AS [Power], MAX([EnginePowerHP]) AS [PowerHP],
COUNT(DISTINCT([GearID])) AS [GearCount],
COUNT(DISTINCT([EngineID])) AS [EngineCount],
COUNT(DISTINCT([EquipmentID])) AS [EquipmentCount],
COUNT(DISTINCT([WinchID])) AS [WinchCount],
COUNT(DISTINCT([OwnershipID])) AS [OwnerCount],
COUNT(DISTINCT([PortID])) AS [PortCount],
COUNT(DISTINCT([OperationID])) AS [OperationCount]
FROM QueryView
WHERE CensusID = @CensusID
GROUP BY VesselID
ORDER BY UniqueRegistrationNumber
```

Since the criteria for the vessel selection is named by the user at runtime, it cannot be added to the query directly. Therefore, a separate *QuerySummaryTableAdapter* partial class is defined within the *QuerySummaryTableAdapter.cs* class, to extend the basic query shown above by adding the WHERE and HAVING clauses reflecting the user criteria specified. This is done through simple string functions to insert the clauses as required, and then the query is run against the database to *QuerySummaryDataTable* object of the business logic class with the data retrieved. Code snippets are shown here:

```
string newCommand = originalCommand.Substring(0, length) + whereClause +
   originalCommand.Substring(length - 1, lengthOrder - length - 1);
if (havingClause != "")
   newCommand += " HAVING " + havingClause.Substring(5);
newCommand += originalCommand.Substring(lengthOrder - 1);
   ...
FleetRegisterBL.VesselSummaryDataSet.QuerySummaryDataTable dataTable = new
   FleetRegisterBL.VesselSummaryDataSet.QuerySummaryDataTable();
this.Adapter.Fill(dataTable);
return dataTable;
```

This data table is returned back to the *VesselCommunications* class, specifically to the *QueryVessel()* method which invoked the database query, depending on the language setting with which the application is configured:

```
public FleetRegisterBL.VesselSummaryDataSet.QuerySummaryDataTable QueryVessel
   (int censusID, string where, string having, bool nationalLanguage)
{
   try
   {
      if (nationalLanguage)
         return tamngVesselSummary.QuerySummaryTableAdapter.GetVesselQuerySummaries
            (censusID, where, having);
      else return
         tamngVesselSummary.QuerySummaryTableAdapter.GetVesselQuerySummariesENG
            (censusID, where, having);
   }
   catch
   {
      return null;
   }
}
```

This method, in turn, forwards the same object up to the user interface level, particularly to the *LoadVesselsGrid()* method in *frmMain*, which retrieves the vessels to load in the result grid according to the criteria which would have previously been specified.

```
dsSummary.QuerySummary.Clear();
if (simpleWhere == "")
   dsSummary.QuerySummary.Merge
      (VesselCommunications.Instance.QueryVessel
      (currentCensusID, queryForm.GetWhereClause(), queryForm.GetHavingClause(),
       Settings.Default.LookupLanguageNational));
else
   dsSummary.QuerySummary.Merge(VesselCommunications.Instance.QueryVessel
      (currentCensusID, simpleWhere, "", Settings.Default.LookupLanguageNational));
```

As seen above, the search criteria may have been defined by the user either within *frmVesselQuery* or *frmVesselSimpleQuery*. Both these forms formulate strings which represent the user criteria as to be used added to the database query, which are transferred across the various application tiers, however the query form is the advanced form allowing for both *WHERE* clauses on all the fields as well as *HAVING* clauses for the tables which are allowed multiple records per vessel, such as engines and gears, whereas the simple query form is restricted to a *WHERE* clause on a few selected fields. These forms are shown according to the menu item click made by the user, the initial use interface actions which trigger the flow of information described here.

Preparation for Client-Server system

Implementation of a client-server structure within the Fleet Register application has not taken place, due to the restrictions that many countries have with both hardware and connectivity. However, preparations have been made in order to enable the change to such a structure without the need for excessive modifications.

The Fleet Register Communications library may simply be replaced by a service, which will be placed on the server side, along with the database, data access layer and business logic. This service would implement all of the methods which are currently present within the communications project (and ideally their interfaces also), and they could be split up according to the functionality, as is currently done.

The following is an example taken from the *CensusCommunications.cs* class, which is part of the *FleetRegisterComm* library. The methods *RenameCensus* is currently implemented as shown below:

```
public bool RenameCensus(int censusID, string censusName)
{
   try
   {
      tamngCensus.CensusTableAdapter.UpdateCensusName(censusName, censusID);
      return true;
   }
   catch
   {
      return false;
   }
}
```

In order to allow for a client-server structure, a new web service, or WCF service, may be created. An interface, named *ICensus.cs* for example, could define the *RenameCensus* method as follows:

```
bool RenameCensus(int censusID, string censusName);
```

The method could then be implemented, within a different class that inherits from this interface, named *Census.cs* for example, in exactly the same way as shown above, initialising the *TableAdapterManager tamngCensus* within the constructor, also as it is presently done.

Within the calling class, which in this case is *frmMain* of the *FleetRegisterUI*, the service will have to be instantiated before use, as *CensusCommunications* is currently used as a static class and therefore not instantiated. Once an instance of the service is created and connected to, the *RenameCensus* method call will then be performed on the service, without its name or the parameters having to be modified.

In this way, the entire system may be split up, and the smart client separated from the server functionality, without major changes. Thus, the effort required to set up a client-server architecture is cut down and the need for in-depth testing from scratch is reduced.

Database Structure

The Fleet Register database is built upon the relational database model and is managed within SQL Server Management Studio Express.

The database is logically divided into two parts: the reference and the register. The reference is a set of look-up tables which provide a list of options used to populate fields within the register. The register contains all the information related to the census and the vessels pertaining to it.

Register Tables

The register is the part of the database which is really concerned with the vessel census. The census is a list of vessels, and each vessel has a vast range of attributes which are logically divided into tables for better organisation.

The Entity Relationship Diagrams may be found on the following pages, followed by an explanation of each table and its fields.

Register tables have been customised according to requirements of particular countries, which may collect more or less information about vessels depending on the census in place and the operations within the country. Presently, the Fleet Register has been customised for use in Albania, Egypt and Lebanon, and the fields added in each case have been marked.

Some fields within the Register tables are only relevant for countries which are members of the EC. These fields are included within all versions in any case; however are disabled and invisible to countries which are not member states, according to application configuration. These fields are also indicated within the list of Register tables.

N.B. – The foreign keys to reference tables which are mentioned are only symbolic, but not included in the database. This is done to allow for null values, which are represented as empty strings.

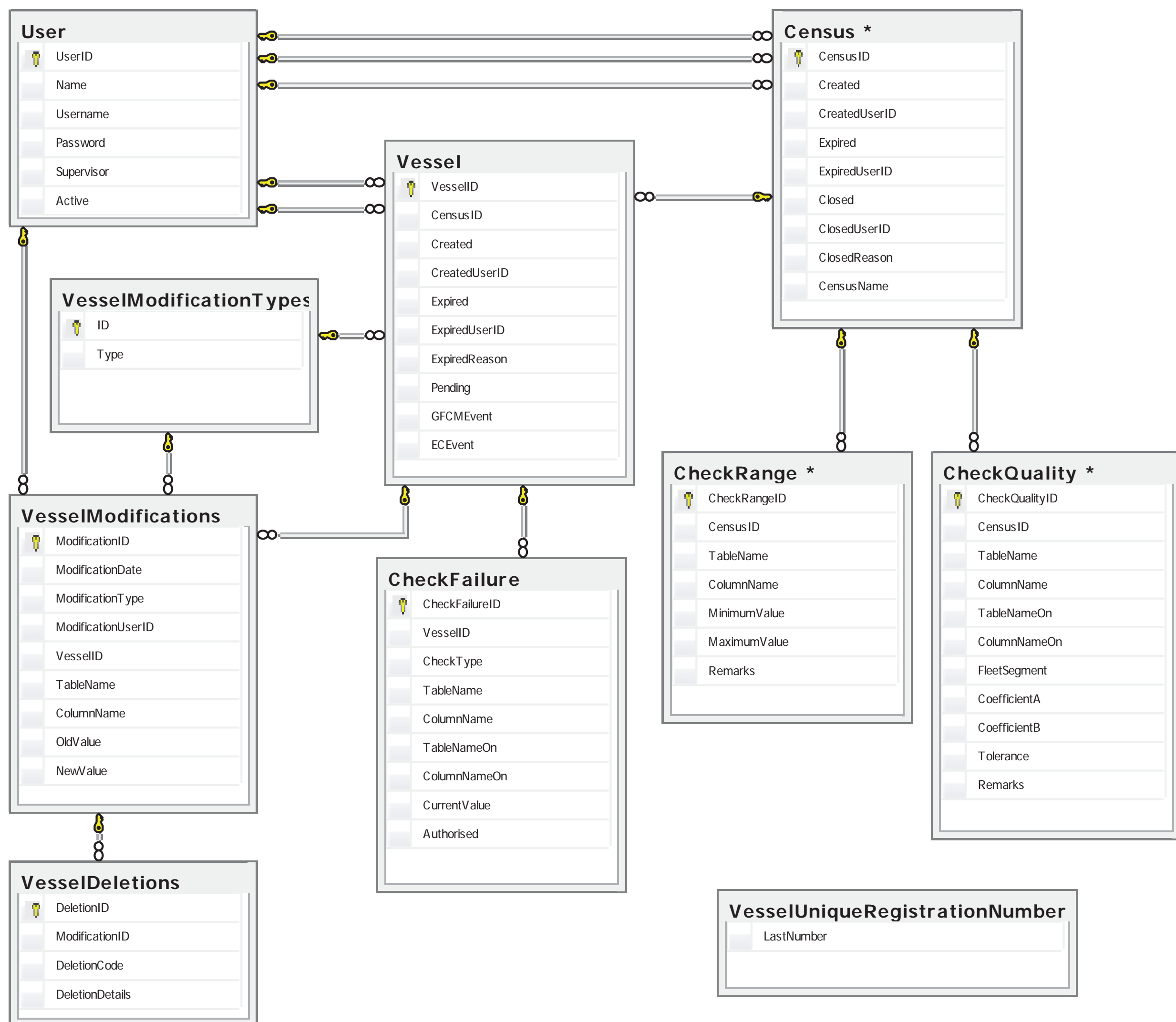


Figure 2 – Register ERD

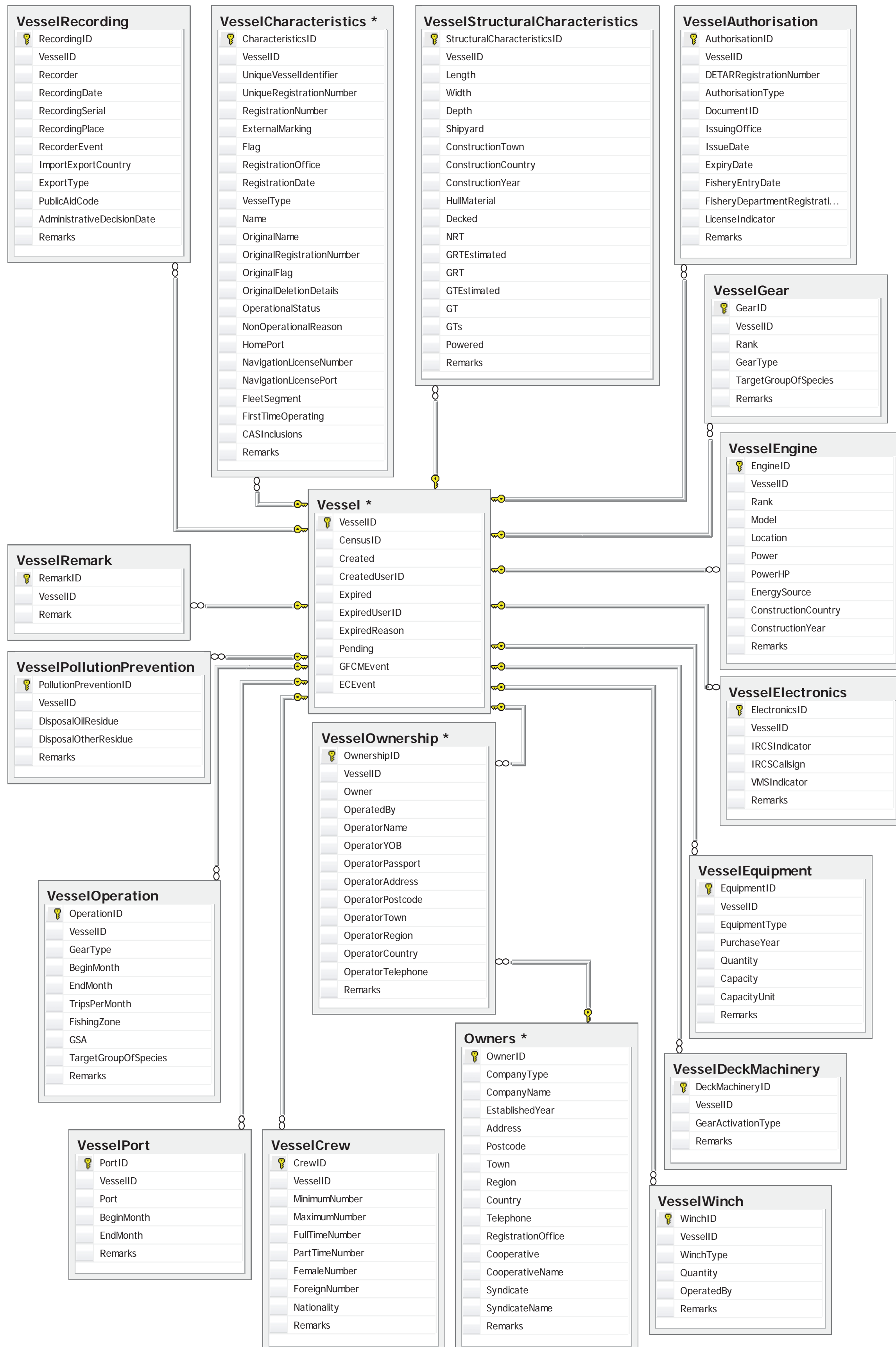


Figure 3 – Vessel ERD

Register Tables Details

User	
This table stores the information about the user and his/her login details.	
Field	**Description**
UserID	automatically generated integer value which acts as the primary key and unique identifier for the record
Name	string representing the name of the user
Username	string of maximum 10 characters representing the username to login to the system
Password	string of maximum 10 characters representing the password required to login to the system
Supervisor	boolean determining whether the user has supervisor rights
Active	boolean determining whether the username and password are active within the system

Table 1 – Users table

Census	
This table stores information about the census and its status.	
Field	**Description**
CensusID	automatically generated integer value which acts as the primary key and unique identifier for the record
Created	datetime value showing when the census was created
CreatedUserID	integer which acts as a foreign key to the Users table and indicates which user created the census
Expired	datetime value showing when the census was deleted
ExpiredUserID	integer which acts as a foreign key to the Users table and indicates which user deleted the census
Closed	datetime value showing when the census was closed
ClosedUserID	integer which acts as a foreign key to the Users table and indicates which user closed the census
ClosedReason	string representing the reason for which the census was closed, as given by the user
CensusName	string representing the name of the census

Table 2 – Census table

Vessel	
This table stores information about the status of the vessel data.	
Field	**Description**
VesselID	automatically generated integer value which acts as the primary key and unique identifier for the record
CensusID	integer which acts as a foreign key to the Census table and indicates which census the vessel information pertains to
Created	datetime value showing when the vessel record was created
CreatedUserID	integer which acts as a foreign key to the Users table and indicates which user created the vessel record
Expired	datetime value showing when the vessel record was modified or deleted
ExpiredUserID	integer which acts as a foreign key to the Users table and indicates which user expired the vessel record
ExpiredReason	integer which acts as a foreign key to the VesselModificationTypes table and indicates which the reason for which the record was expired, as assigned by the system
Pending	boolean representing whether the record has some fields which are to be checked and modified/confirmed by the supervisor
GFCMEvent	boolean indicating whether the event which caused the creation of the vessel record is required for GFCM reporting or not
ECEvent	boolean indicating whether the event which caused the creation of the vessel record is required for EC reporting or not

Table 3 – Vessel table

VesselRecording	
This table stores information about the survey carried out on the vessel.	
Field	**Description**
RecordingID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the recording information pertains to
Recorder	string which acts as a foreign key to the RefRecorders table and indicates which recorder carried out the survey
RecordingDate	datetime value showing when the survey took place
RecordingSerial *(Lebanon)*	integer which indicates the serial number present on the data collection sheet used during the survey
RecordingPlace *(Lebanon)*	string which acts as a foreign key to the RefPorts table and indicates the place where the details related to the vessel where captured during the survey
RecorderEvent	string which acts as a foreign key to the RefEvents table and indicates the event which caused the need for the survey to be modified
ImportExportCountry *(EC)*	string which acts as a foreign key to the RefCountries table and indicates the country that the vessel was imported from or exported to
ExportType *(EC)*	string which acts as a foreign key to the RefExportTypesEU table and indicates the type of vessel export which took place
PublicAidCode *(EC)*	string which acts as a foreign key to the RefPublicAidsEU table and indicates if the vessel owner received public aid in relation to the event being recorded
AdministrativeDecisionDate *(EC)*	datetime value indicating when the decision on confirmation of event was taken by administration
Remarks	string which stores any remarks related to this record

Table 4 – Vessel Recording table

VesselCharacteristics	
This table stores information about the various ways of identifying the vessel, and its most fundamental details.	
Field	**Description**
CharacteristicsID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the characteristic information pertains to
UniqueVesselIdentifier *(Lebanon)*	string representing a unique vessel identifier, of a format which differs from the Unique Registration Number below
UniqueRegistrationNumber	string representing a unique registration number used to identify the vessel within the EC and the GFCM
RegistrationNumber	string representing the national registration number assigned to the vessel
ExternalMarking	string representing the external marking on the vessel
Flag	string which acts as a foreign key to the RefCountries table and indicates the country in which the vessel is registered
RegistrationOffice	string which acts as a foreign key to the RefMinorStrata table and indicates the country in which the vessel is registered
RegistrationDate	datetime value showing when the vessel was registered
VesselType	string which acts as a foreign key to the RefVesselTypes table and indicates the type of the vessel
Name	string representing the name of the vessel
OriginalName	string representing the original name of the vessel, in the case that it has been changed
OriginalRegistrationNumber	string representing the original national registration number of the vessel, in the case that it has been changed
Original Flag	string which acts as a foreign key to the RefCountries table and indicates the original flag where it was registered, in the case that it has been changed

OriginalDeletionDetails	string used to describe the reason for which the vessel was deleted from the original country's database, in the case that it has been changed
OperationalStatus	string which acts as a foreign key to the RefOperationalStatuses table and indicates the operational status of the vessel
NonOperationalReason	string which acts as a foreign key to the RefInactivityReasons table and indicates the reason for which the vessel is not in use, in the case that it is non-operational
HomePort	string which acts as a foreign key to the RefPorts table and indicates the port in which the vessel is registered
NavigationLicenseNumber *(Lebanon)*	string representing the number present on the vessel's navigation licence
NavigationLicensePort *(Lebanon)*	string which acts as a foreign key and represents the port to which the vessel is registered on its navigation licence
FleetSegment	string which acts as a foreign key to the RefFleetSegmentation table and represents the segment to which the vessel pertains, according to the national system
FirstTimeOperating	boolean indicating whether it is the first time the vessel was found to be operating within the country
CASInclusions	string indicating which samples for the CAS survey the vessel has been included in
Remarks	string which stores any remarks related to this record

Table 5 – Vessel Characteristics table

VesselStructuralCharacteristics	
This table stores the information related to the physical attributes of the vessel.	
Field	**Description**
StructuralCharacteristicsID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the structural characteristic information pertains to
Length	decimal value representing the length of the vessel
Width	decimal value representing the width of the vessel
Depth	decimal value representing the depth of the vessel
Shipyard	string representing the shipyard where the vessel was constructed
ConstructionTown *(Lebanon)*	string which acts as a foreign key to the RefTowns table and inidicates the town in which the vessel was constructed
ConstructionCountry	string which acts as a foreign key to the RefCountries table and indicates the country in which the vessel was constructed
ConstructionYear	integer which represents the year in which the vessel was constructed
HullMaterial	string which acts as a foreign key to the RefHullMaterials table and indicates the material with which the vessel was constructed
Decked	boolean indicating whether the vessel has a deck
NRT (Lebanon)	decimal value representing the Net Registered Tonnage of the vessel
GRTEstimated *(Lebanon)*	boolean indicating whether the GRT of the vessel has been estimated or measured precisely
GRT	decimal value representing the Gross Registered Tonnage of the vessel
GTEstimated *(Lebanon)*	boolean indicating whether the GT of the vessel has been estimated or measured precisely
GT	decimal representing the Gross Tonnage of the vessel
GTs *(EC)*	decimal representing the increase in tonnage permitted on grounds of safety
Powered	boolean indicateing whether the vessel is powered by at least one engine
Remarks	string which stores any remarks related to this record

Table 6 – Vessel Structural Characteristics table

VesselAuthorisation	
This table stores the information about the fishing authorisation for the vessel.	
Field	**Description**
AuthorisationID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the fishing authorisation information pertains to
DETARRegistrationNumber *(Albania)*	string representing the DETAR registration number assigned to the vessel
AuthorisationType	string which acts as a foreign key to the RefAuthorisationTypes table and indicates the type of fishing authorisation assigned to the vessel
DocumentID	string representing the identifier of the authorisation document
IssuingOffice	string which acts as a foreign key to the RefIssuingOffices table and indicates the office which issued the fishing authorisation
IssueDate	datetime value showing when the fishing authorisation was issued
ExpiryDate	datetime value showing when the fishing authorisation will expire
FisheryEntryDate	datetime value showing when the vessel entered the fishery department records
FisheryDepartmentRegistrationNumber	string representing the registration number within the fishery department
LicenseIndicator	boolean indicating whether the vessel has a valid fishing license
Remarks	string which stores any remarks related to this record

Table 7 – Vessel Authorisation table

VesselGear	
This table stores the information about the fishing gear the vessel uses. The vessel may include more than one gear.	
Field	**Description**
GearID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the gear information pertains to
Rank	integer representing the priority of the gear
GearType	string which acts as a foreign key to the RefGears table and indicates the particular type of the gear
TargetGroupOfSpecies	string which acts as a foreign key to the RegGroupOfSpecies table and indicates which group of species the vessel is targeting with that particular gear
Remarks	string which stores any remarks related to this record

Table 8 – Vessel Gear table

VesselEngine	
This table stores information about the engine which powers the vessel. The vessel may include more than one engine.	
Field	**Description**
EngineID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the engine information pertains to
Rank	integer representing the priority of the engine
Model	string representing the model of the engine
Location	string which acts as a foreign key to the RefEngineLocations table and indicates the location of the engine aboard the vessel
Power	decimal representing the power of the vessel, in kW
PowerHP	decimal representing the power of the vessel, in HP

EnergySource	string which acts as a foreign key to the RefEnergySources table and indicates the energy source of the engine
ConstructionCountry	string which acts as a foreign key to the RefCountries table and indicates the country in which the engine was built
ConstructionYear	integer representing the year in which the engine was built
Remarks	string which stores any remarks related to this record

Table 9 – Vessel Engine table

VesselElectronics	
This table stores information about the electronic devices installed on the vessel.	
Field	**Description**
ElectronicsID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the electonics information pertains to
IRCSIndicator	boolean indicating whether the vessel has an International Radio Call Sign
IRCSCallsign	string representing the IRCS of the vessel, if available
VMSIndicator	boolean indicating whether the vessel has a Vessel Monitoring System
Remarks	string which stores any remarks related to this record

Table 10 – Vessel Electronics table

VesselEquipment	
This table stores information about the electronic equipment available on the vessel. The vessel may include more than one piece of electronic equipment.	
Field	**Description**
EquipmentID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the electronic equipment information pertains to
EquipmentType	string which acts as a foreign key to the RefEquipment table and indicates the particular type of the electronic equipment
PurchaseYear	integer representing the year in which the equipment was purchased
Quantity	integer representing the number of pieces of equipment of that type
Capacity	decimal representing the capacity of the equipment
CapacityUnit	string which acts as a foreign key to the RefCapacityUnits tables and indicates the unit of measurement for capacity
Remarks	string which stores any remarks related to this record

Table 11 – Vessel Electronic Equipment table

VesselDeckMachinery	
This table stores information about the deck machinery installed on the vessel.	
Field	**Description**
DeckMachineryID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the deck machinery information pertains to
GearActivationType	string which acts as a foreign key to the RefGearActivationMethods table and indicates the type of gear activation method used on the vessel
Remarks	string which stores any remarks related to this record

Table 12 – Vessel Deck Machinery table

VesselWinch	
This table stores the information about the winches available on the vessel.	
Field	**Description**
WinchID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the winch information pertains to
WinchType	string which acts as a foreign key to the RefWinchTypes table and indicates the winch type in use
Quantity	integer representing the number of winches of that particular type
OperatedBy	string which acts as a foreign key to the RefWinchOperationTypes table and indicates the way in which the winch is operated
Remarks	string which stores any remarks related to this record

Table 13 – Vessel Winch table

VesselOwnership	
This table stores the information about the company and owner of the vessel.	
Field	**Description**
OwnershipID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel ownership information pertains to
Owner	integer which acts as a foreign key to the Owners table and indicates the owner or co-owner of the vessel
OperatedBy	string which acts as a foreign key to the RefOperators table and indicates the person who operates the vessel
OperatorName	string representing the name of the operator,if not the owner
OperatorYOB (Lebanon)	integer representing the year of birth of the operator, if not the owner
OperatorPassport (Lebanon)	string representing the passport number of the operator, if not the owner
OperatorAddress	string representing the street address of the operator, if not the owner
OperatorPostcode	string representing the postcode of the operator, if not the owner
OperatorTown	string acting as a foreign key to the RefTowns table, if present, and representing the town of the operator, if not the owner
OperatorRegion	string representing the region of the operator, if not the owner
OperatorCountry	string which acts as a foreign key to the RefCountries table and indicates the country in which the operator is found, if not the owner
OperatorTelephone (Lebanon)	string representing the telephone number of the operator, if not the owner
Remarks	string which stores any remarks related to this record

Table 14 – Vessel Ownership table

VesselCrew	
This table stores information about the crew which works on the vessel.	
Field	**Description**
CrewID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the crew information pertains to
MinimumNumber	integer representing the minimum number of crew members on the vessel
MaximumNumber	integer representing the maximum number of crew members on the vessel
FullTimeNumber	integer representing the number of crew members registered as full-time workers
PartTimeNumber *(Albania)*	integer representing the number of crew members registered as part-time workers

FemaleNumber *(Albania)*	integer representing the number of female crew members on the vessel
ForeignNumber *(Lebanon)*	integer representing the number of foreign crew members on the vessel
Nationality *(Lebanon)*	string acting as a foreign key to the RefCountries table and indicating the nationality of the majority of the foreign crew members on the vessel
Remarks	string which stores any remarks related to this record

Table 15 – Vessel Crew table

VesselPort

This table stores information about the port where the vessel is based. The vessel may include more than one port.

Field	Description
PortID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the port information pertains to
Port	string which acts as a foreign key to the RefPorts table and indicates the port where the vessel is based
BeginMonth	integer representing the first month in which the vessel is based at this port
EndMonth	integer representing the last month in which the vessel is based at this port
Remarks	string which stores any remarks related to this record

Table 16 – Vessel Port table

VesselOperation

This table stores information about the fishing operation which the vessel carries out. The vessel may include more than one operation.

Field	Description
OperationID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the fishing operation information pertains to
GearType	string which acts as a foreign key to the RefGears table and indicates the category in which the gear in use is found
BeginMonth	integer representing the first month in which the gear is used on the vessel
EndMonth	integer representing the last month in which the gear is used on the vessel
TripsPerMonth	integer representing the average number of fishing trips made per month
FishingZone	string which acts as a foreign key to the RefFishingZones table and indicates the fishing zone in which the operation takes place
GSA	string which acts as a foreign key to the RefGeoSubAreas table and represents the GSA in which the operation takes place
TargetGroupOfSpecies	string which acts as a foreign key to the RefGroupOfSpecies table and indicates the group of species targeted by this operation
Remarks	string which stores any remarks related to this record

Table 17 – Vessel Operation table

VesselPollutionPrevention

This table stores information about the safety equipment available on the vessel, and the methods use to prevent pollution.

Field	Description
PollutionPreventionID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the safety information pertains to

DisposalOilResidue	string which acts as a foreign key to the RefDisposalTypes table and indicates the way in which oil residue is disposed of
DisposalOtherResidue	string which acts as a foreign key to the RefDisposalTypes table and indicates the way in which other residue is disposed of
Remarks	string which stores any remarks related to this record

Table 18 – Vessel Pollution Prevention table

VesselRemark	
This table stores any remarks related to the vessel as a whole.	
Field	**Description**
RemarkID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the remark information pertains to
Remark	string representing the general remarks related to the vessel

Table 19 – Vessel Remark table

Owner	
This table stores the information about companies and owners.	
Field	**Description**
OwnerID	automatically generated integer value which acts as the primary key and unique identifier for the record
CompanyType	string which acts as a foreign key to the RefCompanyTypes table and indicates the type of company which owns the vessel
CompanyName	string representing the name of the company or individual
EstablishedYear	integer representing the year in which the company was established or the year in which the individual was born
Address	string representing the street address of the company or individual
Postcode	string representing the postcode of the company or individual
Town	string acting as a foreign key to the RefTowns table, if present, and representing the town of the company or individual
Region	string representing the region of the company or individual
Country	string which acts as a foreign key to the RefCountries table and indicates the country in which the company or individual is found
Telephone *(Lebanon)*	string representing the telephone number of the company or individual
RegistrationOffice	string which acts as a foreign key to the RefMinorStrata table and indicates the port authority with which the company is associated
Cooperative *(Lebanon)*	boolean indicating whether the company or individual is affiliated to a cooperative
CooperativeName *(Lebanon)*	string which acts as a foreign ket to the RefCrewCoops table and represents the the name of the cooperative that the company or individual is affiliated to, if any
Syndicate *(Lebanon)*	boolean indicating whether the company or individual is affiliated to a syndicate
SyndicateName *(Lebanon)*	string which acts as a foreign ket to the RefCrewSyndicates table and represents the name of the syndicate that the company or individual is affiliated to, if any
Remarks	string which stores any remarks related to this record

Table 20 – Owners table

VesselModificationTypes

This table stores the types of modification which may be performed on the vessel record.	
Field	**Description**
ID	automatically generated integer value which acts as the primary key and unique identifier for the record
Type	string representing the type of modification made on the vessel record

Table 21 – Vessel Modification Types table

VesselModifications	
This table stores the details of the modifications which caused the vessel record to be expired.	
Field	**Description**
ModificationID	automatically generated integer value which acts as the primary key and unique identifier for the record
ModificationDate	datetime showing when the modification took place
ModificationType	integer which acts as a foreign key to the VesselExpiryReasons table and indicates which type of modification was performed
ModificationUserID	integer which acts as a foreign key to the Users table and indicates which user modified the vessel record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel was modified
TableName	string representing the name of the modified table
ColumnName	string representing the name of the modified column
OldValue	string representing the original value before modification
NewValue	string representing the new value after modification

Table 22 – Vessel Modifications table

VesselDeletions	
This table stores information about vessels which have been deleted	
Field	**Description**
DeletionID	automatically generated integer value which acts as the primary key and unique identifier for the record
ModificationID	integer which acts as a foreign key to the VesselModifications table and indicates which vessel the deletion pertains to
DeletionCode	string which acts as a foreign key to the RefEvents table and indicates the event which caused the need for the vessel to be deleted
DeletionDetails	string which stores any remarks related to the deletion of the vessel

Table 23 – Vessel Deletions table

VesselUniqueRegistrationNumber	
This table stores information about the sequential number to be used when generating the unique registration number for the vessel	
Field	**Description**
LastNumber	integer value which represents the last number which was used when generating the vessel unique registration number

Table 24 – Vessel Unique Registration Number table

CheckRange

This table stores information about the range checks which will be performed on the vessels within the census	
Field	**Description**
CheckRangeID	automatically generated integer value which acts as the primary key and unique identifier for the record
CensusID	integer which acts as a foreign key to the Census table and indicates which census the check is to be performed for
TableName	string which indicates the table within which the field to be checked is found
ColumnName	string which indicates the field to be checked
MinimumValue	integer representing the minimum acceptable value for the field
MaximumValue	integer representing the maximum acceptable value for the field
Remarks	string which stores any remarks related to this record

Table 25 – Check Range table

CheckQuality	
This table stores information about the quality checks which will be performed on the vessels within the census	
Field	**Description**
CheckQualityID	automatically generated integer value which acts as the primary key and unique identifier for the record
CensusID	integer which acts as a foreign key to the Census table and indicates which census the check is to be performed for
TableName	string which indicates the table within which the field to be checked is found
ColumnName	string which indicates the field to be checked
TableNameOn	string which indicates the table within which the field to be compared to is found
ColumnNameOn	string which indicates the field to be compared to
FleetSegment	string which acts as a foreign ket to the RefFleetSegmentation table and indicates which fleet segment the vessel must be in for this check to be applied
CooefficientA	decimal representing the 'a' coefficient for the linear regression to be checked
CooefficientB	decimal representing the 'b' coefficient for the linear regression to be checked
Tolerance	integer representing the tolerance margin for this check
Remarks	string which stores any remarks related to this record

Table 26 – Check Quality table

CheckFailure	
This table stores information about the failed checks which were performed on vessels	
Field	**Description**
CheckFailureID	automatically generated integer value which acts as the primary key and unique identifier for the record
VesselID	integer which acts as a foreign key to the Vessel table and indicates which vessel the check was performed on
CheckType	string which indicates whether the check was a Range or Quality check
TableName	string which indicates the table within which the field which was checked is found
ColumnName	string which indicates the field which was checked
TableNameOn	string which indicates the table within which the field which was compared to is found, if a Quality check was performed
ColumnNameOn	string which indicates the field which was compared to, if a Quality check was performed
CurrentValue	string which shows what the checked value was
Authorised	boolean indicating whether the failure was authorised to result in a valid vessel

Table 27 – Check Failure table

Reference Tables

The reference tables within the fleet register are used in order to list the possible options for a number of fields within the register. The reference is fairly static; although users are given the possibility to modify and update the tables, they are likely to remain unchanged.

These options are drawn up according to the national system and the FAO guidelines, but they may also be extended to include GFCM and EC guidelines.

Basic Table Structure

The basic tables within the reference database have the following fields:

Field	Description
ID	string which acts as the primary key and unique identifier for the record
Description	string representing the description of the option in relation to the list
Name	string representing the name of the option in national language
NameENG	string representing the name of the option in English

Table 28 – Basic reference table

N.B. – The ID field is set as a string instead of an integer in order to retain the codification system, for example in the RefGeoSubAreas table where some areas are assigned codes such as 11.1, 11.2, or the RefEventsEU table where events are identified by textual codes such as CEN, EXP, MOD, etc.

The following are the tables which follow this schema and are used within the Fleet Register application:

Table	Description
RefAuthorisationTypes	Stores the list of possible fishing authorisation types which the vessel may be assigned within the country
RefCapacityUnits	Stores the list of possible units of capacity for vessel equipment
RefCompanyTypes	Stores the list of company types which may own the vessel
RefCrewSyndicates (Lebanon)	Stores the list of Syndicates that owners or crew members may belong to
RefDisposalTypes	Stores the list of ways in which waste from the vessel may be disposed of
RefEnergySources	Stores the list of energy sources which may be used to power the engine on the vessel
RefEngineLocations	Stores the list of positions on the vessel where the engine may be placed
RefEquipmentCategories	Stores the list of categories which the list of vessel equipment may be divided into
RefEventClassesEU	Stores the list of event classes that the events may be categorised into according to the EU regulations
RefExportTypesEU	Stores the list of different export types which the EU recognises
RefFAOSubAreas	Stores the list of largest divisions of the Mediterranean according to FAO
RefFishingTechniquesEU	Stores the list of fishing techniques that fishing gears may be linked to, according to the EU
RefFishingZones	Stores the list of different zones in which vessels of the particular country are allowed to fish
RefGearActivationMethods	Stores the list of ways in which vessel gears may be activated
RefGearClasses	Stores the list of classes which the list of vessel gears may be divided into
RefGearsEU	Stores the list of vessel gears according to the EU
RefGroupOfSpeciesGFCM	Stores the list of species groups according to the GFCM
RefHullMaterialsEU	stores the list of materials which the vessel hull could be constructed from

	according to the EU
RefInactivityReasons	Stores the lift of reasons for which the vessel may be inactive within the fleet
RefIssuingOffices	Stores the lift of offices which may issue the vessel with a fishing authorisation
RefMarineFisherySubsectors	Stores the different fleet sectors to which fishing vessels may pertain within the particular country
RefMonths	Stores the names of months in the year
RefOperationalStatuses	Stores the list of statuses
RefOperators	Stores the different entities which may be in charge of the operation of the vessel
RefPortTypes	Stores the list of the different tyoes of ports which may be present within the country
RefPublicAidsEU	Stores the list of different types of public aid schemes offered by the EU
RefRecorders	Stores the list of names of people working as recorders to gather vessel data
RefStrata	Stores the list of teh largest geographical divisions within the country
RefTowns (Lebanon)	Stores the list of towns within the country
RefVesselTypesFFFAO	Stores the list of vessel types according to the FAO Fishing Fleet
RefWinchOperationTypes	Stores the list of ways in which the vessel winches may be operated
RefWinchTypes	Stores the list of types of winch which may be used on the vessel

Table 29 – List of basic reference tables

The following is a list of reference tables which follow the basic structure but are not used within the Fleet Register and are included only for completeness sake and in the case of application extension in the future.

Table	**Description**
RefFisherySectors	Stores the list of different types of fishery sectors that the vessel may form part of within the country
RefFishPresentationTypes	Stores the list of different fish presentation types
RefFishTags	Stores the list of possible fish tags according to fish size
RefGearUnits	Stores the list of gear units which may be used to calculate effort
RefIceFormats	Stores the lift of formats which the ice plant on the vessel may take
RefLogbookNAReasons	Stores the list of possible reasons for not having a valid logbook on board the vessel
RefPermanentModifications	Stores the list of possible reasons for modifying a vessel record permanently

Table 30 – List of basic reference tables currently not in use

Extended Tables

Some reference tables include extra fields, which are used to link the tables between them, often creating a hierarchical structure, or to translate the options according to different guidelines.

The following is a list of all the tables which do not follow the basic structure, and a description of the additional fields:

RefCountries	
This table stores the list of countries in the world	
Field	**Description**
NameFRN	string representing the French name of the country
NameESP	string representing the Spanish name of the country
UN	string representing the UN identifying code of the country
UNDP	string representing the UNDP identifying code of the country
ISO2	string representing the 2-letter ISO code of the country
ISO3	string representing the 3-letter ISO code of the country
GFCM	boolean indicating whether the country forms part of the GFCM
EU	boolean indicating whether the country forms part of the EU
MedFisis	boolean indicating whether the country forms part of the MedFisis project area

Table 31 – Countries reference table

RefCrewCoops ***(Lebanon)***	
This table stores the list of cooperatives which the owners or crew members may be affiliated with	
Field	**Description**
Syndicate	string which acts as a foreign key to the RefCrewSyndicates table

Table 32 – Crew Coops reference table

RefEquipment	
This table stores the list of equipment possibly available on the vessel	
Field	**Description**
EquipmentCategory	string which acts as a foreign key to the RefEquipmentCategory table

Table 33 – Equipment reference table

RefEvents	
This table stores the list of possible events which would cause and addition, modification or deletion of a vessel record	
Field	**Description**
EventClassEU	string which acts as a foreign key to the RefEventClassesEU table
EventCodeEU	string which acts as a foreign key to the RefEventsEU table

Table 34 – Events reference table

RefEventsEU	
This table stores the list of possible events which would cause and addition, modification or deletion of a vessel record, according to EU regulations	
Field	**Description**
EventClassEU	string which acts as a foreign key to the RefEventClassesEU table

Table 35 – EU Events reference table

RefFAOStatisticalDivisions	
This table stores the list of geographical divisions of the Mediterranean for statistical purposes, according to FAO	
Field	**Description**
FAOSubArea	string which acts as a foreign key to the RefFAOSubAreas table

Table 36 – FAO Statisticl Divisions reference table

RefFleetSegmentation	
This table stores the list of Fleet Segments that a vessel may pertain to, as defined by the GFCM	
Field	**Description**
AlphaCode	char which represents the segment identifier according to the GFCM

Table 37 – Fleet Segmentation reference table

RefGears	
This table stores the list of gears according to the international ISSCFG list	
Field	**Description**
MatchingCodeGear	string which is currently not in use
GearClass	string which acts as a foreign key to the RefGearClasses table
GearEU	string which acts as a foreign key to the RefGearsEU table
GearSTD	string which represents the international standard abbreviation of the gear
FishingTechniqueEU	string which acts as a foreign key to the RefFishingTechniquesEU table
MarineFisherySubsector	string which acts as a foreign key to the RefMarineFisherySubsectors table

Table 38 – Gears reference table

RefGearsNational	
This table stores the list of gears customised according to the national system	
Field	**Description**
GearClass	string which acts as a foreign key to the RefGearClasses table
Gear	string which acts as a foreign key to

Table 39 – Gears reference table

RefGeoSubAreas	
This table stores the list of Geographical Sub Areas	
Field	**Description**
FAOStatisticalDivision	string which acts as a foreign key to the RefFAOStatisticalDivisions table

Table 40 – Geographical Sub Areas reference table

RefGroupOfSpecies	
This table stores the list of species groups	
Field	**Description**
GroupOfSpeciesGFCM	string which acts as a foreign key to the RefGroupOfSpeciesGFCM table

Table 41 – Group of Species reference table

RefHullMaterials	
This table stores the list of materials which the vessel hull could be constructed from	
Field	**Description**
HullMaterialEU	string which acts as a foreign key to the RefHullMaterialsEU table

Table 42 – Hull Materials reference table

RefLOAClasses	
This table stores the list of possible classes of the vessel Length Overall	
Field	**Description**
MinLength	decimal representing the minimum length for the class
MaxLength	decimal representing the maximum length for the class
Decked	boolean indicating whether the vessel is decked

Table 43 – LOA Classes reference table

RefMinorStrata	
This table stores a list of the smallest geographical divisions within the country	
Field	**Description**
Substratum	string which acts as a foreign key to the RefSubstrata table

Table 44 – Minor Strata reference table

RefPorts	
This table stores the list of ports in the country	
Field	**Description**
PortCode	string which represents the national code assigned to the port
MinorStratum	string which acts as a foreign key to the RefMinorStrata table
Latitude	string which represents the latitude of the port
Longitude	string which represents the longitude of the port
GeoSubArea	string which acts as a foreign key to the RefGeoSubAreas table
PortType	string which acts as a foreign key to the RefPortTypes table
PortCodeEU	string which represents the EU code assigned to the port

Table 45 – Ports reference table

RefSpecies	
This table stores the list of fishable species	
Field	**Description**
NameSCI	string representing the scientific name for the species
NameFRN	string representing the French name for the species
NameESP	string representing the Spanish name for the species
ISSCAAP	string representing the ISSCAAP group that the species belongs to
TAXOCODE	string representing the code assigned to the species within the international taxonomic database
ALPHACODE	string representing the international 3-letter code assigned to the species
Author	string representing the author who first documented the species
Family	string representing the family of the species
Order	string representing the order of the species

Table 46 – Species reference table

RefSubstrata	
This table stores a list of the medium-level geographical divisions within the country	
Field	**Description**
Stratum	string which acts as a foreign key to the RefStrata table

Table 47 – Substrata reference table

RefVesselTypes	
This table stores the list of vessel types according to the international ISSCFV list	
Field	**Description**
STDAbbreviation	string representing the international standard abbreviation of the vessel type
VesselTypeFFFAO	string which acts as a foreign key to the RefVesselTypesFAO table

Table 48 – Vessel Types reference table

RefVesselTypesNational

This table stores the list of vessel types customised according to the national system	
Field	**Description**
VesselType	string which acts as a foreign key to the RefVesselTypes table

Table 49 – Vessel Types National reference table

Utility Tables

There are a few tables within the reference system which do not follow the basic structure at all, but have a completely different set of columns. These tables are mainly used to carry out utility functions such as filtering, group identification or data integrity checks.

The following is the set of utility tables and a description of the structure of each:

RefFleetSegmentIdentification	
This is a table used to identify the GFCM fleet segment of a vessel, given its main gear, target group of species for the gear and length overall	
Field	**Description**
FleetSegment	string which acts as a foreign key to the RefFleetSegmentation table
GearType	string which acts as a foreign key to the RefGears table
TargetGroupOfSpecies	string which acts as a foreign key to the RefGroupOfSpecies table
MinLength	decimal which indicates the minimum length of vessels which qualify for the fleet segment
MaxLength	decimal which indicates the maximum length of vessels which qualify for the fleet segment

Table 50 – Fleet Segment Identification reference table

RefGearGroupOfSpecies	
This is a table used to define the acceptable groups of species for each gear, in order to allow for filtering	
Field	**Description**
GearType	string which acts as a foreign key to the RefGears tables
GroupOfSpecies	string which acts as a foreign key to the RefGroupOfSpecies table

Table 51 – Gear-Group of Species reference table

RefTDistribution	
This is a table listing the parameters used to calculate confidence intervals as part of statistical reporting (please refer to the following for further details: http://oreilly.com/catalog/transqlcook/chapter/ch08.html)	
Field	**Description**
p	decimal which represents a coefficient that corrects the standard error around the mean according to the size of a sample
df	int which represents the degrees of freedom value

Table 52 – T-Distribution reference table

RefVesselGearTypes	
This is a table used to define the acceptable gears for each vessel type, in order to allow for filtering	
Field	**Description**
VesselType	string which acts as a foreign key to the RefVesselTypes table
GearType	string which acts as a foreign key to the RefGears table

Table 53 – Vessel-Gear Types reference table

Fleet Register Configurator

The Fleet Register Configurator is a separate application used to apply user preferences and settings to the Fleet Register. It is run independently from the Fleet Register, automatically upon installation, and also has its own menu item inserted into the Start menu on installation so that it may be launched when necessary to modify the settings.

The Fleet Configurator is divided into a number of sections, each of which will be described in the following sections.

Country-related Settings

The country-related settings are a set of values which customise the installed application according to the particular location of installation and the user preferences. The values to be set are the following:

- EU Country: boolean indicating whether the country is a member state of the EC
- Country: list selection identifying the country in which installation is taking place
- Vessel Unique Identifier Prefix: string representing the first 3 letters for system-generated CFR numbers, mainly used within the EC
- National Language – References: boolean indicating whether the reference lookups will show national language text or English text
- National Language – Labels: boolean indicating whether the field labels will be shown in the national language or English
- Skip Integrity Checks: boolean indicating whether the integrity checks will be left out of the application, possibly for performance reasons or because the country has not identified the check criteria
- Peripheral Office: boolean indicating whether the installation is taking place at a peripheral office, in which case some options, such as deletion of a vessel, will be disabled

All the values are stored within the *FleetRegisterUI* settings, so that they can easily be accessed within the Fleet Register application. The values are loaded automatically into the fields by setting the *ApplicationSettings EditValue* of each field in the designer, and saving a change to either value is simply done when the *Apply Changes* button is clicked, as follows, taking the Vessel Unique Identifier Prefix as an example:

```
Properties.Settings.Default.UniqueRegNoPrefix = this.txtUIDPrefix.Text;
```

SQL Server Parameters

This section contains just a couple of values to allow connection to the database, namely:

- SQL Server Address
- SQL Instance Name

These may be changed to connect to a different database, with a *Test Database Settings* button to ensure that a connection is present and functional. These values are accessed and changes in the same way as the country-related settings above.

Reference Table Configuration

The Reference Table Configuration sections deals with the initialisation of reference tables which are known to be highly country-specific and therefore set by the administrators of the system before the application begins to be used. The tables are specifically the following:

- Recorders
- Authorisation Types
- Issuing Offices
- Fishing Zones
- Strata
- Substrata
- Minor Strata
- Ports

All the tables are loaded directly from the database to start off, according to the connection data set at design-time. The user is given the freedom to modify these tables as necessary, including the links between the various tables, such as Strata, Substrata and Minor Strata, adding and removing records if required. The tables are then updated using the *FleetRegisterConfiguratorDS* data set's automatically generated table adapter manager, with the *UpdateAll()* method.

Labels Configuration

This section allows for the users to translate the labels used within the application into the national language, as pre-set as design-time.

All the labels are stored within an XML file called *Translations.xml*, in the Language subfolder defined in the application settings, within the user's *My Documents* folder. Each item within the XML file contains an ID element to identify the label, an English language label, a label for each national language (so far Albania, Arabic for Lebanon and Arabic for Egypt) and a placeholder for the label to be shown within the application.

Upon loading the Fleet Register Configurator, the entire XML file is loaded into a data set reflecting the XML schema, and the English and national language columns displayed (currently only one national language at a time). The user may modify the labels as required, but may not add or remove any. Upon save, the data set is written to overwrite the existing *Translations.xml* file, which will them include all the updated label data.

Existing Database and Language Translations

The Fleet Register Configurator caters for re-installation of the Fleet Register application on a machine which already had a previous version installed. In order to prevent the loss of register data and language translations, the user is given the option to retain the files currently in use, as opposed to using the base version which are included with the installation package.

The installation version of the files are installed in the *DatabaseInstallation* and *LanguageInstallation* subfolders within the application folder in *My Documents*, which are different from the subfolders defined for use by the application, by default the *Database* and *Language* subfolders.

If the user has no previous existing database, the file within the installation folder will be copied to the database folder. Otherwise, if the user decides to retain the existing version of the database, no changes to the file system will be made; whereas if the user would like to use the newly-installed database, a new folder is created to archive the existing database, and then the new version is copied to the database sub folder, overwriting the current version in use. This process, of course, may only be carried out if the database structures of the new and existing databases are exactly the same. This is a check which must be carried out by the developer when deciding whether to enable this functionality in the Configurator when a new version of the software is released.

In the case of the language files, it is slightly more complex than simply copying files, in the case that the existing file is to be retained, as the elements within the XML files are analysed to check whether new labels have been added. Therefore, both the installation and existing version of the Translations.xml file are loaded into data sets. Each element in the new file is checked, so that if it does not exist within the current file, that element is appended. At the end of this process, the existing file will contain any new elements, although not yet translated, and this extended data set is written to XML to overwrite the existing version. The cases when there is no existing language file or the user does not wish to retain the existing one follow the same process as the database files above.

This functionality gives the user the possibility of installing new versions of the software without loss of information, provided that the structures are the same. This, in turn, gives the developer greater flexibility in bug-fixing and releasing new versions without hindering the work of users who have already put the software into use.

Developer's How-To Guide

This section of the document is intended as a step-by-step guide to performing a number of tasks which are common in the enhancement and customisation of the Fleet Register. Basic knowledge of the structure and operation of the application is assumed, so one is advised to have read the previous section of the document beforehand.

Adding Fields

One of the most common requirements during customisation of the application for a particular country is the addition of fields which are related to the vessel required at a national level. The following is a guide to how this should be carried out. Code snippets are not generally included, as the source code may be consulted and the location of the code is specified throughout.

1. Update the Database

Tables

The table to which the field is to be added must be identified. The field must be added to the table and, like all other fields, be set not to allow nulls.

In the case that the field will be a linked to a reference table, the data type is to be set to *nvarchar(50)*, which is the standard data type for the identifiers within the reference tables. The reference table is to be set up according to the basic reference structure as shown on page 24, with any additional fields required. No foreign key is required, to cater for the case in which the field is left blank and no reference ID is present.

In the case that an entire table needs to be added, it is important to remember to include an integer primary key field to the table, which will be an automatically incremented value, as well as an integer foreign key to the *Vessel* table, named *VesselID* for consistency's sake.

Views

In order for the field to be included in the vessel query, it must be added to the *QueryView*. If the table is already existent, then the field simply has to be added to the *SELECT* list. Otherwise, a *LEFT OUTER JOIN* is required between the Vessel table and the added table. Note that if the table allows multiple records per vessel, it is important to add criteria to join only when the fields of the table are not all blank.

There are also some views that need to be extended for reporting purposes. If the field forms part of a multiple entry table, then the appropriate view must have its *SELECT* list extended, for example the *EnginesView* and *EngineENGView* if the field is part of the *VesselEngine* table. Otherwise, the field is to be added to the *FullFleetView* or *FullFleetENGView*. If the field is one of the main identifying vessel fields and therefore is to be included in the Small Fleet Report, the *SmallFleetView* and *SmallFleetENGView* are also to be extended. Note that these views include *LEFT OUTER JOIN*s to the reference tables, and the *Name* or *NameENG* field is to be shown depending on whether it is an *ENGView* or not.

If the field is to be included in an EU or GFCM report, the *EUFleetView* and *GFCMFleetView* views are to be considered and extended in the same way. In this case, pay special attention to reference tables and the field which is required to satisfy the reporting requirements. Joining to EU or GFCM specific tables may be necessary.

2. Extend the Data Access Layer

Vessel Fields

The data sets in the data access layer, *FleetRegisterDA*, must be extended to reflect the database changes. The first step is adding the field, or possibly table, to the *VesselDataSet*. Apart from the field being added to the table, the table adapter must also be updated. All three queries, namely *GetData*, *Insert* and *SelectByVesselID*, must have the field added to their field list. Remember to check that the *ExecuteMode* for the *Insert* query remains set to *Scalar* once the query SQL is modified.

In the case that the added field is an important vessel identifying field, and therefore must be displayed on the vessel summary list within the application, the *VesselSummaryDataSet* must be updated, and the field added to both the *VesselSummary* and *QuerySummary* tables. All the *GetSummaries* queries must also be changed.

Both when downloading/uploading vessels or exporting the full fleet, all vessel fields are considered. Therefore, the related data sets, *TransferDataSet* and *FullExportDataSet*, must also be expanded. Note that each table adapter within the *FullExportDataSet* has two queries which need to be updated: the *Readable* version requires all used reference tables to be joined, whereas the *Reference* version does not. Remember also to check that the namespace of the *FullExportDataSet* is not automatically changed, since the files are found within the *Export DataSets* subfolder of *FleetRegisterDA*.

Reference Tables

In the case that reference tables are added to cater for the new fields, the tables and their table adapters must be added to the *ReferenceDataSet*, to allow the linking to the field within the application, and also to the *ReferenceMgmtDataSet*, to allow for user modification of the tables. Remember to follow the naming convention for the queries within the *ReferenceDataSet*.

Notes

In order to ensure that the table adapter manager for each table adapter is automatically generated, ensure that the table adapter has a definition for each of the following within its properties: *DeleteCommand*, *InsertCommand*, *SelectCommand* and *UpdateCommand*. The *(New)* option may be selected if either of these is missing.

Once all the changes to the data access layer are made, ensure that the *DataSet* project property for each of the data sets is set to *FleetRegisterBL*, so that all the related business layer objects will be created or updated as soon as the project is built.

3. Modify the Communications Classes

Since the communications layer is mainly an encapsulation of the data access layer, not many modifications are required at this point.

Within the *VesselCommunications* class, the *Insert* method for the table where the fields were added must be updated, to ensure that the value to be inserted is passed as a parameter to the data access level query.

If any reference tables have been added, the *ReferenceCommunications* class must be extended. The connection to the table adapters of the new table within the *ReferenceDataSet* and *ReferenceMgmtDataSet* must be initiliased and set within the constructor. Four methods must be added per new table, namely: *GetBasicRef()*, *GetRefNameByID()*, *GetRefNameENGByID()* and *ExistsRef()*, all of which are direct calls to queries within the data sets. Also, the *SelectReferenceMgmtTables()* method must have the new tables included and filled, and the *UpdateReferenceTable()* method must have a new case statement added for the new table. An *UpdateRef()* method must also be created, to call the *Update* query on the table.

Any new reference tables must also be included in the reference export, so the *ExportCommunications* class must be modified. The table adapter connection must be initialised within the constuctor, as above. The *SelectReferenceExport()* method must have a command added to include the new table and fill it.

4. Revise the User Interface Forms

Vessel Details

The main form which needs to be revised in the case of addition of vessel fields is *frmVesselDetails*. A control must be added into the correct *LayoutControl* on the appropriate *TabControl*, depending on which table the field would have been added to. The type of control depends on the data type of the field:

- a *TextEdit* is required for a string
- a *SpinEdit* is required for any numeric value
- a *CheckEdit* for a boolean value
- a DateEdit for a date
- a LookUpEdit for a link to a reference field

In each case, the *DataBindings EditValue* must be set to link to the field within the *BindingSource* for the vessel table, to automatically load data when present.

In the case that the field is a lookup, a binding source for the reference table must be added, the *DataSource* property set to *dsReference*, which contains all reference tables, and the *DataMember* set to the appropriate table. Then, the lookup's *DataSource* property must be set to the name of the newly-created binding source, the Columns collection must be modified to hide the *ID* column, remove the *Description* column and show the *NameENG* and *Name* columns. The *ValueMember* must be set to the *ID* column, and the *Display* member to *NameENG* (this is eventually modified through code depending on the user preferences).

In the case that the field is numeric, there are validation methods which may be set to ensure that it is a positive number, for example *spnPositiveNumber_EditValueChanging()* and *spnPositiveNumber_Keypress()*. The same goes for past dates or years. The entire set of methods may be found in the *Validation* region, within *Form Events*, within *Methods*.

If the field is part of a multiple entry table, it must be added through the appropriate *GridView* designer. A *ColumnEdit* control of the correct type must be created, as explained above, within the *Columns* menu, and the *FieldName* property must be set to match the field name within the data set table. The layout may also be modified to change the order and width of the columns.

Within the code, a number of modifications must also be made as follows:

- *Loading* region
  - The label of the field must be set within the *LoadFormStrings()* method
  - If a reference table has been added, the language to be used must be set within the *LoadLookupsLanguage()* method and the data loaded within *LoadReferenceData()*
  - If the field is numeric, the minimum and maximum values must be set within the *LoadDataEntryRestrictions()* method
  - If the field is only valid for EU member states, add code to potentially hide the field in the *LoadECFields()* method
  - The field must be added to the *LoadVesselData()* method
- *Form Events* region
  - Any logic, such as enabling/disabling fields according to selected user values or events linked to the grid views, must be placed here
- *Data* region
  - If the field is compulsory, the logic to validate it must be included in the *Validate()* method for the appropriate tab
  - The *CheckNewModifications()* and *CheckModifications()* methods must be extended to check for changes in the new field
  - The relevant *GetTable()* method must be extended to save the data input by the user to the business logic object, or the default value, if left blank
  - The *GetModificationsTable()* method and *GetModificationsENGTable()* must have the new fields added in, to record the user changes made

Vessel Summaries

For those vessel fields which were added to the *VesselSummaryDataSet* tables, a modification to the user interface layer which reflects the change is also necessary. The forms which must be modified are: *frmMain*, *frmpendingVessels* and *frmDeletedVessels*.

In each case, there is a *grdvwSummary* object which is to be considered. Through the designer, the new field must be added as column and the *FieldName* set to the name of the field in the data set. The *ColumnEdit* property does not need to be set in this case, as the grids are read-only and the reference values already converted to the correct strings.

Code-wise, the column caption must be set in the *LoadFormStrings()* method, but no other changes are necessary.

Querying Vessels

The advanced vessel query allows the user to set criteria on any vessel field. Therefore the *frmVesselQuery* form will always have to be modified. This is the hardest form to work with, due to the number of controls it contains, so patience is required.

For each new field, three controls must be added to the relevant layout control:

- a *CheckEdit* which determines whether the field is to have any search criteria
- a *CheckEdit* which indicates whether the field criteria is that it has a value and is not blank
- another control, or two in the case of numeric values or dates, within which to specify the search criteria, which will depend on the data type and may require validation or linking to a reference table as explained above in the Vessel Details section

Within the code, a number of modifications are needed, specifically:

- *Loading* region
  - The label of the field must be set within the *LoadFormStrings()* method
  - If a reference table has been added, the language to be used must be set within the *LoadLookupsLanguage()* method and the data loaded within *LoadReferenceData()*
  - If the field is numeric, the controls are to be initialised within *LoadInitialData()*
  - If the field is only valid for EU member states, add code to potentially hide the field in the *LoadECFields()* method
  - The field must be added to the *LoadVesselData()* method
- *Form Events* region
  - A *CheckChanged* event is required for the first *CheckEdit* of each field, to enable or disable the criteria fields and *NotBlank CheckEdit* according to whether criteria will be set or not
  - A *CheckChanged* event is required for the *NotBlank CheckEdit* of each field, to disable the criteria fields in case it is selected by the user
  - In the case of dates, an *EditValueChanged* event is needed for the *From DateEdit*, in order to ensure that to *To* date is greater or the same
  - In the case of numeric fields, an *EditValueChanged* event is required for the *From* and *To SpinEdit*s, to ensure that the minimum is less than or equal to the maximum
  - The *Reset* button *Click* event must be extended to reset the newly-added field
- *Data* region
  - Code is to be inserted into the *GetWhereClause()* method to add the user specified search criteria to the field, if any
  - Only in the case that an entire multiple records table is added, the *GetHavingClause()* method would have to be extended to add criteria for the count of records of that table linked to the vessel

The simple query form contains a subset of the fields to query, and is therefore a restricted version of the advanced query. The exact same procedure as above is to be repeated on the *frmSimpleQuery* in the case that the newly-added field is considered to be an important query field.

5. Extend the Validator Utility

The *Validator.cs* class contains a number of methods required to check the vessels once a full census scan takes place. The *Census Validation* region is the one to be modified, in particular the *Validate()* method for the table to which the fields were added. A check is required:

- for compulsory fields, to ensure they are not blank
- for fields linked to reference tables, to ensure that the value specified exists within the reference table
- for date validation such as checking if they are in the future or in the past
- for dependencies between fields, for example ensuring that the registration date cannot be before the construction date
- other custom validation, such as checking that the unique registration number is unique amongst the vessel records

In the case that a new multiple records table is added, a new method must also be added to the *Blank Tables* region, to check whether the table is blank or contains only a single default row.

Having completed this step, the process to add a field would be complete. The same code would have to be checked in the case that fields were to be removed, or maybe have their data type modified. Although a long process, the instructions should serve as a more detailed look into how the application works and how the code is structured.

Customising Reference Tables

The user/administrator of the system is given some degree of flexibility in the management of reference files, in that the Configurator provides the functionality to customise some of the tables completely and the application allows for the modification of the textual fields of all the tables. However, when customising the application for a new country, there is some ground-work which may be done beforehand to simplify the process

Setting National Tables

The *RefGears* and *RefVesselTypes* tables store the internationally recognised gears and vessel types, according to the ISSCFG and ISSCFV lists. However, these classifications may not give the level of detail which a country requires, or may not map directly to the categories in use nationally. Therefore, the *RefGearsNational* and *RefVesselTypesNational* provide a way to define categories for use at a national level, whilst linking to the internationally-recognised classification for reporting purposes.

The *RefGearsNational* table starts out as a copy of the *RefGears* table, and the same for the vessel types. Then, rows within the *National* table may be added, deleted or modified as necessary. It is important to ensure that each row in the *National* table retains its link to a row in the original table, through the *Gear* or *VesselType* field. Ideally, a comprehensive numbering system should be used, to immediately indicate the original gear or vessel type from which the national one is derived, since the links will not be visible to users within the application, except through the *Reference Management* tool.

These tables are not visible within the Fleet Register Configurator to avoid system administrators from making modifications once the system is already in use, as that may cause problems given that the vessel type and gears are important information for the vessel.

Migrating Data

In the case that the application is being customised for a country that already had a previous version of the Fleet Register in place, some care with regards to the references is needed when migrating the existing data to the new system.

In the case that the reference tables are not used within any international reports, then the *ID* column values of the reference records may be changed to match the existent ones, in order to simplify migration. However, if the codes adhere to some standard and are used when exporting data externally, the IDs must be retained and a system for mapping the old and new codes devised. This may be done either by drawing up a mapping table and developing a small routine to translate the IDs once when the migration takes place, or by inserting an intermediate table. Examples of intermediate tables are the *RefEvents*, which provides more options than the *RefEventsEU* table, which is used for reporting to the EC, and *RefGroupOfSpecies*, which further divides the *RefGroupOfSpeciesGFCM* categories which are defined for reporting to the GFCM. These tables may be taken as examples.

Customising the Visual Aspect

Small changes to the user interface and the outputs can go a long way in satisfying the users' need for ownership of the application. All country-specific images that are used throughout the application are contained within a single folder which is installed within the local file system in order to allow access to the user, to be able to modify or add images as necessary.

The folder, which at design-time is the *Images* folder within the *FleetRegisterReports* module, will be installed in the user's personal data folder, as explained in the Content Files section of Creating an Installation Package, on 54. Within this folder, the following items should be placed:

- The country's flag, which should be called *Flag.jpg*, prefixed by the country's ISO3 code, for example *MLTFlag.jpg* for Malta. This will be displayed in the application's main form.
- The country's letterhead to be displayed at the top of reports of portrait orientation, called *MLT.jpg* for example, where *MLT* is to be replaced by the country's ISO3 code.
- The country's letterhead to be displayed at the top of reports of landscape orientation, called *MLTLandscape.jpg* for example, where *MLT* is to be replaced by the country's ISO3 code.

The application is designed to show text or nothing at all, in the case that the images are not found. Therefore, the installation package may be built without these images and they can be added to the *ImagesSubFolder* specified in the settings file at a later stage.

Modifying Integrity Checks

Although the data checks are not an aspect of the software which is considered at the initial phase of customisation, but once a certain familiarity with the application is obtained, they become a potential area for improvement in the quality of data. Here we will consider both options of amending or adding to existing checks, as well as creating a new kind of quality control, in this case giving an example of checking whether certain values are far from the fleet average.

1. Insert the Data into the Database

The tables in the database which are to be considered in the case of existing data integrity checks are the *CheckRange* and *CheckQuality* tables, the first providing a test of whether a value falls within a certain range and the second determining that two vessel characteristics have a dependency on each other that follows a linear regression, according to the fleet segment of the vessel.

The existing records in the table may be modified or deleted, or new records may be added, as required. It is important to note that all checks whose *CensusID* is set to 0 will be copied to all new censuses which are created within the database.

In the case that a new type of check is required, a new table must be set up. For the given example, it could be something like the following:

- *CheckFleetCharacteristics* table
    - *CheckCharacteristicID*: an automatically incrementing integer identifier
    - *CensusID*: an integer acting as a foreign key to the Census table
    - *TableName*: a string identifying the table to be checked
    - *ColumnName*: a string identifying the column to be checked
    - *Function*: a string determining what fleet function, in SQL, will be taken
    - *Tolerance*: an integer indicating what percentage of difference is allowed

Therefore, if the required check was to ensure that the vessel length is no more than 50% greater than the fleet average, the following information would be inserted:

- *CensusID*: 0
- *TableName*: VesselStructuralCharacteristics
- *ColumnName*: Length
- *Function*: AVG
- *Tolerance*: 50

Ideally, the table which stored the failures, the *CheckFailure* table, would also be updated. The *Function* column should also be added there, in order to give full information on which check would have failed, given that different functions could be used on the same field.

2. Update the Data Access Layer

All the data access layer objects related to the quality control checks are found within the *CheckDataSet*.

For the existing tables, since the changes made to the database tables are only new data, not modified structure, no changes need to be made to the data set. For the new type of check, on the other hand, a new table must be added to reflect the new database table added, and the table adapter must be created with the following queries:

- *ExistsCheck*: to see if any check in the database matches the *CensusID*, *TableName*, *ColumnName* and *Function*
- *GetCheck*: retrieve a check by *CensusID*, *TableName*, *ColumnName* and *Function*
- *GetChecksByCensusID*: get all checks for a *CensusID*
- *InsertCheck*: insert a new check passing all data as parameters except for the automatically-generated identifier

The *CheckFailure* table and its table adapter also have to be updated according to the change made in the database.

3. Extend the Communications Class

Since the business logic objects would be updated automatically when building, no major changes are necessary within the *CheckCommunications* class in the *FleetRegisterComm*, when existing checks are modified.

For the new controls, the only changes to the existing code would be to the *InsertCheckFailuresWithoutCheck()* and *InsertCheckFailures()* methods, in order to cater for the new field, and to *InsertChecksCopy()*, to copy over the newly-developed fleet characteristics checks when a new census is created from an existing one. A number of new methods would also need to be implemented to follow the current structure:

- *Select* region
  - *SelectFleetCharacteristicChecks*: retrieve all checks and return a *CheckFleetCharacteristicDataTable* of the automatically-generated *FleetRegisterBL.CheckDataSet* class
  - *SelectFleetCharacteristicChecksByCensusID*: retrieve all checks for the census whose ID is passed as a parameter and return a *CheckFleetCharacteristicDataTable* of the automatically-generated *FleetRegisterBL.CheckDataSet* class
  - *SelectFleetCharacteristicCheck*: retrieve any check matching the table, column, function and census passed as parameters and return a *CheckFleetCharacteristicDataTable* of the *FleetRegisterBL.CheckDataSet* class
  - *ExistsFleetCharacteristicCheck*: return a boolean indicating whether a check exists that matches the table, column, function and census passed as parameters
- *Update* region:
  - *UpdateFleetCharacteristicChecks*: call the *CheckFleetCharacteristic* table adapter's *Update* query to update the entire table with the *FleetRegisterBL.CheckDataSet. CheckFleetCharacteristicDataTable* passed as a parameter

4. Add to the User Interface

The first addition which needs to be made to the user interface is a new form which follows the same structure of *frmRangeChecks* and *frmQualityChecks*. It will show a grid, with one column per database column, apart from the automatic identifier. Each column will have the appropriate *ColumnEdit* required for the user to edit the values, and will have its *FieldName* set to the column in the table in the data access layer. Apart from the methods to load the form and the data, and the grid view events which need to be handled, the main method required is that which will call the *Update* method in the communications class as explained above, in order to update the checks in the database. A button must also be added within the *References* > *Data Integrity Checks* menu to launch this form.

In the case that a column would have been added to the *CheckFailure* table, the *frmCheckFailures* form must also have its grid extended to cater for the new field.

In *frmMain*, within the *Methods*, *Option Clicks*, *Fleet* region, there is the *DataIntegrityCheck()* method which begins the actual census scan. Although this makes use of the valiator, which will be dealt with next, the code which retrieves the checks for validation must be extended to cater also for the new ones. The same goes for the *CheckPendingRangeQualityChecks()* method within *frmVesselDetails*.

5. Extend the Validator Utility

The validator within the *FleetRegisterUtilities* is where the data integrity checks are actually run, and therefore where the most important change must take place.

The region to be considered is the *Data Integrity Checks* region. Here the following extensions are required:

- Implement *CheckRangeQualityFleetCharacteristicChecks()* as the modified version of the *CheckRangeQualityChecks()* method, to accept an extra parameter which is the list of fleet characteristic checks to run and to perform the check by, for example, calcuating the fleet average of the length, checking whether the tolerance level is passed, and inserting a *CheckFailure* in that case
- Implement a new method, *CheckPendingFleetCharacteristicChecks()*, which will scan a single pending vessel and check whether any fleet characteristic checks still fail and therefore the vessel should remain pending

These are the methods which will be invoked in the *frmMain* and *frmVesselDetails* forms, as mentioned above, and their implementation is possibly the hardest part of the implementation.

As can be seen from the above process, catering for a new type of quality control check requires some thought and work, and modifications throughout the application tiers. However, a simple modification to the existing set of checks does not require intervention from the developer, and this is why the user is given the option to implement such changes through the forms launched by clicking the *Range Checks* and *Quality Checks* items in the *Reference* > *Data Integrity Checks* menu.

Modifying Consistency Checks

Currently, the data consistency checks are built-into the system, and not database-driven like the data integrity checks examined above. Therefore, addition, deletion or modification of any of the checks requires code changes.

The modifications required have already been explained in the Adding Fields section, namely by modifying the *Validate()* methods in the *Data* section of the vessel details form in the user interface (page 36) and by extending the *Validator* utility's *Census Validation* region (page 39). Kindly refer to these sections for further details.

Adding Reports

Reports are the best way to get a visual overview of the fleet information stores within the Fleet Register. However, since there is a large amount of information available, it is hard to predict all the possible results which may be of interest to a user. Therefore, it may be necessary to add reports based on the particular requirements of a country. To provide a concrete example, the example will be creating a report to show Distribution by Home Port and Number of Female Crew Members.

1. Ensure Database View covers Requirements

Since the report would be considered a national statistical report, the database view to be reviewed is *StatisticalReportsView*. For the example, the fields needed would be the home port and number of female crew members. All the *PortID*, *PortName* and *PortNameENG* are present; however the *FemaleNumber* from the *VesselCrew* table is not. Since the *VesselCrew* table is already included in the view through a *LEFT OUTER JOIN* with the *Vessel* table, the *FemaleNumber* field must simply be added to the *SELECT* list. Had the table not been part of the view, *the LEFT OUTER JOIN* on VesselID would have had to be added. In the case of fields linked to reference tables, remember also to add the reference table using a *LEFT OUTER JOIN* and include the *ID*, *Name* and *NameENG* fields to the *SELECT* list.

2. Create a Results Table in the Data Access Layer

Within the data access layer, a data table and table adapter are required in order to translate the many records in the database view into results which may be used within the report. These may be added to the *StatisticalReportsDataSet* in *FleetRegisterDA*'s *ReportDataSets* subfolder.

The table required could be called *DistributionByPortAndFemaleNumber* and would require fields and a table adapter query resembling those of *DistributionByVesselTypeAndCrew*, substituting the vessel type for the port, and the crew for the female number. The query will get a list of all ports where females are present and join it to a full list of ports, to calculate how many vessels there are in each port, how many vessels in each port have female crew members and how many female crew members there are in each port.
A totally different SQL query may be necessary in order to create a report which does not follow the same structure. In that case, it is important to remember always to include the values for the entire fleet, just in case the values being considered are not compulsory, as otherwise vessels not reporting the field will be completely excluded from the statistics.

It is also important to add a partial class for the new table adapter within the *RestrictedStatisticalReports.cs* file in *FleetRegisterDA*. Following the same example as those already implemented, this gives the possibility of generating the report for a subset of the fleet, by appending an appropriate WHERE clause. The same method is used as shown in the Selection of Information section on page 8, kindly refer to this section and the partial classes already present.

3. Extend the Communication Class

The *ReportCommunications* class within the *FleetRegisterComm* module must be extended in order to encapsulate the newly-added query and give access to the results table.

First of all, the new table adapter must be initialised within the class constructor, and it must have its connection set. Then, within the Select region, two simple wrapping methods must be included to call the query in the data access layer and return a business layer object of *FleetRegisterBL.StatisticalReportsDataSet.DistributionByPortAndFemaleNumber* type. One method will take only the *CensusID* as a parameter, to report for all vessels within the census, and the second will also take a filter string, to add criteria dynamically, as mentioned above, and report on a subset of the fleet.

4. Draw Up the Report

All the reports which are used within the application are drawn up using *DevExpress.XtraReports* package, and are found within the *FleetRegisterReports* component. Creating a new report requires adding a report there.

The closest report to the one in the example is once again the *rptCrewDistributionByVesselType*. Looking at the rdlc (or report definition file) of this report gives a clear idea of what the end result will be. The developer is free to choose any layout, using charts, tables and textboxes to display the data. The report may be split into *subreports* if required, but the important thing to note is that the data will always be placed in the *Detail* section.

The report must have an instance of the *FleetRegisterBL.StatisticalReportsDataSet* from which to read the data. This must be set to be the *DataSource* property of the report and the *DataMember* has to be set to the correct table. Calculated fields may also be easily added by using the *Expression Editor*, if necessary, and formatting rules may be added to particular controls.

Code behind may also be added, most importantly to load the data into the data set through the communications method, in the constructor. In those reports which have been implemented, there is also a *LoadFormString()* method to show the labels in the correct language and a *pctHeader_BeforePrint()* event handler which attempts to load the country's letterhead from the Images folder specified in the application settings and shows a textual header if that fails. These methods should be included in any new report.

Since there is a very wide range of functionality that may be implemented in reports, it would be greatly advisable to get familiar with the *XtraReports* package before implementing new reports.

5. Provide Access to the Report through the User Interface

Once the report is prepared within the *FleetRegisterReports* package, a way for the user to access it must be provided. As with the rest of the reports, the most straightforward way is to include a menu item in the *Reporting* > *Statistical Reports* submenu, on the *frmMain* form, and handle its *ItemClick* event to create an instance of the report from the *FleetRegisterReports* module and make it visible to the user.

Adding Export Routines

Export routines are mainly used to send the data to regional commissions, however their purpose is not only restricted to this as they provide a simple way of gathering all or part of the data within the system and outputting it at once in a format that is easy to process. Here, rather than giving a single concrete example, te document will explain the general steps required to create a new Excel or XML export routine within the system.

1. Optionally Create a Database View

In the case of a flat export data set (a single table) with many fields, it is recommended that a database view is created to join all the data as necessary and avoid very complex queries within the application. These views may also link register tables directly to reference tables for a more readable representation of the data, and are similar to those previously mentioned for querying and reporting. Implemented examples which may be referred to are the *GFCMFleetView* and *EUFleetView*.

If the export is split up into multiple Excel sheets or XML elements, then it would be preferable to extract the data directly from the relevant tables, and this step may be skipped.

2. Insert a New Data Set

When it comes to export routines, it is important that the data set in the data access layer (ideally in the *Export DataSets* subfolder of *FleetRegisterDA*) reflects the structure required for the export file, in order to avoid further processing later on. Therefore, if it is a flat file which is required, the new data set will contain a table, or tables, with at least one query to select data directly from the database view, as in the case of *GFCMExportDataSet*. For more complex export routines, on the other hand, the data set may contain multiple tables, each of which independently gathers data from a different view or database tables, as is the case in *GFCMTask1ExportDataSet*. These will then have business logic objects of the same structure automatically created within the *FleetRegisterBL* component.

One should note that the most important step to creating an export routine is having a clear idea of which data is required and how it will be represented. Creating the correct objects in the data access layer, and populating them in a way which does not require further processing on the result, will definitely save a lot of effort.

3. Extend the Communication Class

Similarly to the previous example of creating a new report, in the case of a new export routine the *ExportCommunications* class of the *FleetRegisterComm* module must be extended simply to encapsulate the data access query and return an object of the business logic layer which is populated with the retrieved results. As can be seen from all the methods currently present in the *Select* region, no logic is required at this level other than filling the required data tables with data according to the developed SQL queries.

4. Generate the Export File within the User Interface Layer

At the user interface level, menu items are required to invoke the generation of the export file, and possibly a form may be shown to gather user preferences on the type of file which is to be created and the way in which the data is to be represented. The *frmExportOptions* form is one to look at as an example.

The export creation is actually carried out within the code of *frmMain*, in the *Methods*, *Option Clicks*, *Exporting* section. There are three types of exports covered here, to an Excel file, to an XML file and to CSV (comma-separated value) file. The XML export is the easiest one, as it is carried out directly by the business logic object. The data table objects provide a built-in method called *WriteXML()*, which simply has to be passed the file path, as selected by the user through a standard *SaveFileDialog*. The other exports cannot be done directly from the data set or data table, so a hidden grid is added to the main form and the data is bound to it, in the background. Using the related grid view object, the *ExportToText()* method is called, passing the ‘,’ character as a separator in a *TextExportOptions* object as a parameter, in order to create a CSV file. For an Excel export, it is the same data grid view’s *ExportToXls()* or *ExportToXlsx()* method which may be used, depending on the version required. A more complex example where each table is exported to a separate Excel worksheet may be seen within the *brbtnExportFull_ItemClick()* method, in the *Fleet Exports* section. The gird view also provides method to export to different formats such as html or pdf, so these may also be examined if necessary.

Note that, unless disabled through the Fleet Register Configurator, the census is scanned for data integrity errors before an export is generated to be sent to international authorities. This is to avoid exporting erroneous or inaccurate data. Note also that in this case, the user is given the option of creating a sealed copy of the census. This is simply for audit reasons, so that a snapshot of the data at the point in time that the data was sent may be retained for future reference.

Adding a Language

The Fleet Register is built in a multi-lingual fashion, in order to cater for the different countries in which it is to be used. Localisation, which refers to the possibility of changing the language of the user interface, is made possible through an XML file which stores the translations of each label seen throughout the application. So far, preparation has been made for Albanian, Arabic for Egypt and Arabic for Lebanon, apart from the default language of English. This section describes the steps to be taken to add another language, which is a difference process from the previous ones seen, in that it is not database-driven but file-driven.

1. Insert the Translation in the XML file

The *Translatons.xml* file is found within the *LanguageInstallation* folder in the *FleetRegisterUI*. This is one of the files that get copied over to the local file system upon installation, as will be explained in the *Creating an Installation Package* section. Within the *Language Data Set* root element, there is a node, named *TranslationsTable*, for every string which needs to be translated. This contains a child element for each language being used, each of which is labelled according to the language code, as listed here: http://msdn.microsoft.com/en-us/library/ms533052(v=vs.85).aspx, as well as an empty *LocalName* element, the use of which will be explained shortly.

Therefore, to add a new language into the file, the language code must be identified, and a new child added to those already present. To start off, in case the translation is not available, a copy of the English text is set as a default.

2. Update the Language Data Set

The *FleetRegisterDA* module contains a *LanguageDataSet* with a single table that reflects the structure of the XML file. A field, with the same name as the XML element, that is the language code with any punctuation removed, must be added to this table. Since there is no table adapter in this case, the update of the data access layer requires no more work.

3. Set the Fleet Register Configurator

Since the choice of language is specified by the user within the *Fleet Register Configurator*, in the *Country-Related Settings* section, the option to select the new language must be added. Currently, the language is hard-coded into the system, according to the country for with the application was customised. In the *btnApplySettings_Click()* method, the language setting is set to English, or 'en-GB', if the labels are not set to the national language, or otherwise to 'sq', 'ar-LB' or 'ar-EG', depending on the country. In certain countries, it may be more appropriate to have a drop-down list with multiple languages to select from. This would require a small change in the Configurator, as will be described in the following section.

No further changes are needed within the application in order for the new language to be put to use, so the next subsection will give a general overview of the way in which the translations are loaded at run-time to show the correct language.

General Overview of Localisation System

The culture and language of the application are set upon construction of *frmMain*, in the *LoadSettings()* method. The *Language* value is read from the application's *Settings* file, where it would have been stored by the Fleet Register Configurator. The *CurrentUICulture* and *CurrentCulture* are set to a new instance of *CultureInfo* using this language value, in order to ensure that the whole application follows the correct format for decimal values, dates, etc. Then, a new instance of the *LanguageCommunications* class is instantiated.

The constructor of the *LanguageCommunications* class receives as parameters the path to the Translations XML file, and the *Language* value. It reads the XML file into a *LanguageDataSet* object, and then sets the *LocalName* column to point to the required language column, by setting its *Expression* property to the appropriate language code (once again with punctuation removed).

```
public LanguageCommunications(string path, string local)
{
   dsLanguage = new FleetRegisterBL.LanguageDataSet();
   dsLanguage.ReadXml(path);
   dsLanguage.TranslationsTable.LocalNameColumn.Expression =
      local.Replace("-", "");
}
```

This means that every record will have its *LocalName* value that is the same as the string within the column whose name matches the language code, and changing the *LocalName Expression* instantly changes the language of the *LocalName* string value.

The *LanguageCommunications* class provides a single *GetString()* method, which accepts a string identifier as a parameter. It returns the string found in the *LocalName* column of the record whose *ID* column matches the parameter, or a blank string if it is not found.

```
public string GetString(string id)
{
   try
   {
      return (dsLanguage.TranslationsTable.FindByID(id).LocalName);
   }
   catch
   {
      return "";
   }
}
```

Therefore, all that is required thoughout the user interface is that, whenever a string is to be displayed, the *GetString()* method is called on the *LanguageCommunications* object, passing the ID of the label as a parameter, as below.

```
this.Text = languageCommunications.GetString("strFleetRegister");
```

No changes are required in the user interface to change the language of the application.

As can be seen here, this system of localisation requires slightly more effort at design time, in order to specify all strings within the XML file and ensure that all controls have their text set correctly. However, it gives much more flexibility later on in that the XML files may be translated separately from the application and then included within the system, and the language may be changed with minimal effort.

Adding a Setting to the Configurator

The Fleet Register Configurator may be extended to provide user flexibility in the customisation of any aspect of the application, to the extent of hiding and showing vessel fields, changing consistency checks, etc. However, the most common requirement is the addition of a single configuration setting within the *Country-Related Settings* section, which will be described here.

1. Add the Value to the Settings File

Once the particular configuration value has been identified, it must be added to the *Settings.settings* file which is found in the Properties folder of *FleetRegisterUI*. The name and type must be specified, along with the initial value, and the *Scope* property must be set to *User*, in order to allow changes through the Configurator.

2. Update the Configurator Interface

Access to the new setting must be provided to the user through the Fleet Register Configurator, which is designed as a wizard. In the case of *Country-Related Settings*, the first section of the second wizard page is to be considered. The type of control required depends on the data type of the setting, but any control may be added as long as the value can easily be read out of it. The *ApplicationSetting EditValue* property of the control must be set to the name of the setting within the *Settings* file, to automatically load the value into the control.

3. Save the Setting

A number of steps have to be taken to add the new settings to the flow of the Configurator, within the *frmFleetRegisterConfigurator* code:

- Add a global variable, prefixed by 'original', to store the value of the setting upon load
- In the *SaveOriginals()* method of the *Loading* region, set the *original* variable to the current value upon load
- In the *btnApplySettings_Click()* event handler in the *Events*, *Button Clicks* region, add the code to update the property setting and update the *original* variable to the saved value, or reset to the original value if the saving was unsuccessful
- In the *wizardControl1_SelectedPageChanging()* event handler, add a condition to check whether the value of the settings has been changed but not saved when the second wizard page is changing, in order to alert the user

Whenever a new configuration value is added to the application, consideration must always be made as to whether the user should have access to modify the value, or at least view it for diagnostic purposes, through the Fleet Register Configurator. The important thing is to ensure that the user has no need to browse through the application directory in search of the actual configuration file.

Creating an Installation Package

Once changes have been made to the application package, a new version of the software is required for distribution. The deployment project, named *FleetRegisterSetup*, creates a Microsoft Windows Installer, or MSI Installer for the Fleet Register. This section points out the main features which one should be familiar with to create a new installation package.

1. Primary Outputs

The primary output of the solution will be the executable which will be run by the user. In this case, since *FleetRegisterUI* is set to be the *StartUp Project*, it is the primary output of this project which is required as the primary access point.

There are actually two separate applications which the user may run: the Fleet Register and the Fleet Register Configurator. Therefore, by viewing the *File System* of the *FleetRegisterSetup* project, we can see that under the *User's Program Menu*, a *FAO MedFisis* folder will be installed, and within it two shortcuts to the primary output. The Fleet Register Configurator shortcut specifies '/CONFIG' in the *Arguments* property, whereas the MedFisis Fleet Register specifies none. The *Arguments* property is examined within the *FleetRegisterUI's Program.cs*, the first to be run, to determine which form is to be run. If the '/CONFIG' argument is found, the *frmFleetRegisterConfigurator* form is run, otherwise it will be *frmMain*, to access the application normally.

2. Content Files

Apart from the executable file, there are a number of other types of file which are used by the application and need to be installed locally, specifically:

- The *Translations.xml* file found within the *LanguageInstallation* folder of *FleetRegisterUI*
- The database files found in *FleetRegisterDA*
- The entire *Documents* subfolder of *FleetRegisterUI*
- The entire *Images* subfolder of *FleetRegisterReports*
- The *Ref_Countries.xml* file in the *Configurator* subfolder of *FleetRegisterUI*

For each of these files, the *Build Action* property must be set to *Content* and *Copy to Output Directory* must be set to *Copy always* (the file will be overwritten if it already exists). Within the *FleetRegisterSetup* project, the content files for each of the required modules have been added to the project output list. Viewing the *File System* once again, we can see that wherever the files are within subfolders, the content files are placed directly within the *MedFisis* > *Vessel Register* subfolder of the *User's Personal Data Folder* (commonly the *My Documents* folder), whereas a subfolder named *DatabaseInstallation* is created within the file system for the database files which are not contained in a subfolder within the *FleetRegisterDA* component. This retains a consistent file structure to store the files locally.

It is important to ensure that the latest version of all of these files is included before the installation package is built, paying special attention to the *FleetRegister.mdf* and *FleetRegister_log.ldf* database files which need to be extracted from the SQL Server *Data* folder in the case that database changes have been made.

3. Custom Actions

Upon installation of the Fleet Register, the Fleet Register Configurator is to run automatically in order to configure the application for first use. A custom action is required to prompt the launch of the Configurator upon installation. This is done by viewing the *FleetRegisterSetup Custom Actions*. The primary output is added as a custom action to the *Commit* phase, and the *Arguments* property is set to '/CONFIG' to divert to the Fleet Register Configurator. This avoids having to instruct the user to run the Configurator manually through the *Start* menu after installation.

4. Prerequisites

In order for the application to run successfully, there a few prerequisites which are require, namely:

- .NET framework 3.5 SP1
- SQL Server 2005 Express Edition SP2
- Windows Installer 3.1

These must be selected through *Properties* on the context menu of the *FleetRegisterSetup* project, by clicking the *Prerequisites* button. The option *Download prerequisites from the same location as my application* must be selected in order to include the installers for each within the setup package.

5. Setup Properties

The final step to creating the installation package is setting the correct properties for the *FleetRegisterSetup*. These include *Author*, *ProductName*, *Title* and, most importantly, *Version*, amongst others. Remember to have sequentially incrementing version numbers to be able to trace releases.

Once all this is set, change the *Solution Configuration* to *Release*, rather than *Debug*, and compile the entire solution, including the *FleetRegisterSetup*. This will generate an installation package within the *Release* subfolder of the *FleetRegisterSetup* source code folder. It will include five parts:

- Installation folder for the .NET framework
- Installation folder for SQL Server Express
- Installation folder for Windows Installer 3.1
- setup.exe
- FleetRegisterSetup.msi

The entire set may then be distributed as the full installation package, and instructions given to run *FleetRegisterSetup.msi* in order to install and start using the MedFisis Fleet Register.

Further Comments or Queries

We would like to perfect and keep the MedFisis Fleet Register up-to-date. We know that, although much effort has been put into development and debugging, the best way to properly test and allow the application to progress is by putting it to use in a real work environment. Therefore, we appreciate any feedback highlighting issues within the software or suggestions for improvement, and we also welcome any queries related to customisation or development of extensions of enhancements.

Please feel free to contact us using the standard diplomatic channels, or through the MedFisis website: http://www.faomedfisis.org/

References

Coppola, S., Mosteiro, A. and Camilleri M. 2011. MedStat 2011 – Fishing Vessel Census; Census design and implementation. GCP/INT/918/EC/MedFisis - MedFisis Technical Document, 95 pp.

Coppola, S., Mosteiro, A. and Camilleri M. 2011. MedStat 2011 – Fishing Vessel Census; Operational Manual. GCP/INT/918/EC/MedFisis - MedFisis Technical Document, 39 pp.

MedStat 2011 – Fishing Vessel Census; Operational Manual – Supplement 0. GCP/INT/918/EC/MedFisis - MedFisis Technical Document, 46 pp.

MedStat 2011 – Fishing Vessel Census; Fishing Vessel Register Software: User Guide. GCP/INT/918/EC/MedFisis - MedFisis Technical Document (in preparation).

Pilgrim, D., Coppola, S., Mosteiro, A. and Camilleri M. 2011. MedStat 2011 – Fishing Vessel Census; Training Manual. GCP/INT/918/EC/MedFisis - MedFisis Technical Document (in preparation).

Microsoft MSDN Library
http://msdn.microsoft.com/en-us/library/default.aspx

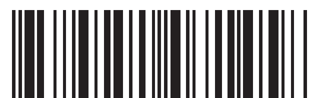
DevExpress Online Documentation
http://documentation.devexpress.com

GFCM Data & Information Reporting Requirements
http://151.1.154.86/gfcmwebsite/DataInformationReportingRequirements.html

Commission Regulation (EC) No 26/2044 on the Community Fishing Fleet Register
http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32004R0026:EN:NOT

The present document is part of a series of documents providing guidance on the implementation of a Fishing Fleet Census, as the technical documentation which goes hand in hand with the MedStat Fishing Vessel Register Software. It provides guidance to developers and technical experts who would like to understand the way in which the software package is designed and built and how it should be managed. Many practical and procedural details are included, in order to give a wide knowledge base to those who intend to customise the program for use in a particular country or situation, or those who would like to develop extensions or enhancements for the application.

ISBN 978-92-5-107145-8



9 789251 071458
I2592E/1/01.12