Programmer's Guide

Climsoft Version 3.0

March 2011

Contents

Introduction to the Programmer's Guide

Purpose of this guide: This guide describes how to modify the Climsoft system to provide new or improved functions to support the requirements of your organisation. The guide is a aimed at experienced programmers, and details the technical processes required to to change or enhance the programs and data structures within the system. We assume that the functional requirements of the modifications have already been specified by the Systems Administrator. Guidelines for specifying these requirements are given in the Climsoft Systems Administrator's Guide.

Skills assumed: In this guide we assume hat you are familiar with using the Microsoft Windows, Excel, Access and Visual Basic (VB) products, including the use of Visual Basic for Applications (VBA) which provides the programming interface to Excel and Access. We also assume that you know how to produce HTML files, which are used as the basis for Climsoft's help system.

ClimSoft Data Model

The ClimSoft data model was first designed at the CLICOM Workshop help at ACMAD, Niger in 1999. The fundamental design principle is to achieve easy and flexible manipulation of data and derivation of products. Observation data are stored in their most elemental form in one table called **observation**. New types of observations can be included easily without the need to modify the database design. See the Climsoft Database Entity Relationship Diagram.

In this model, flexibility is achieved at the expense of disk storage space. However disk storage capacity is no longer an issue given the current trends indicating ever-increasing capacity of storage media.

Each entity (or table) in the model has a single primary key and in this regard, the design takes advantage of the MS Access facility of generating auto-number fields for primary keys. Furthermore, every table has an index called **identification** which ensures uniqueness of records. For example in the observation table, the **identification** index is made up of the fields r*ecorded_from, described_by, classified_into, recorded_at, made_at* ,which will uniquely identify an observation record.

The following section presents (***in bold italics***) brief references to the entities that constitute the main components of the Climsoft data model. We present these as the bare minimum, and we encourage database managers to feel free to extend this basic structure to include details that are specific to their operating environments -- provided the core elements are left intact for compatibility reasons.

The core of the Climsoft data model is the regular ***observation*** made in a ***station***, at ***scheduled*** times. An observation comprises of 3 quantities: the actual observed value, a quality control flag that tells us more about that value, and the observation period over which that value applies. The model has allowance to record special phenomenon e.g. sandstorms, including the moments when they start and when they end, etc.

To be meaningful we need to store the description of the observed ***element*** and a description of the ***instrument*** used. For quality control reasons we encourage the practice to keep a list of elements that a station is capable of recording. Preferably this list of ***station specific elements*** should be prepared before any data ingestion is done for that station, so that those that are not applicable are easy to trap.

The data model is also designed to keep track of physical ***features*** around a station that are likely to influence the element values observed –e.g. a road construction, nearby trees being cut, a new building is put up etc.

The use of auto-number fields is an issue requiring careful attention if the ClimSoft database is to be ported to database management systems which do no have provision for generating auto-numbers e.g. MS SQL Server. This issue is discussed under setting up a back-end database in the Administrator's Guide.

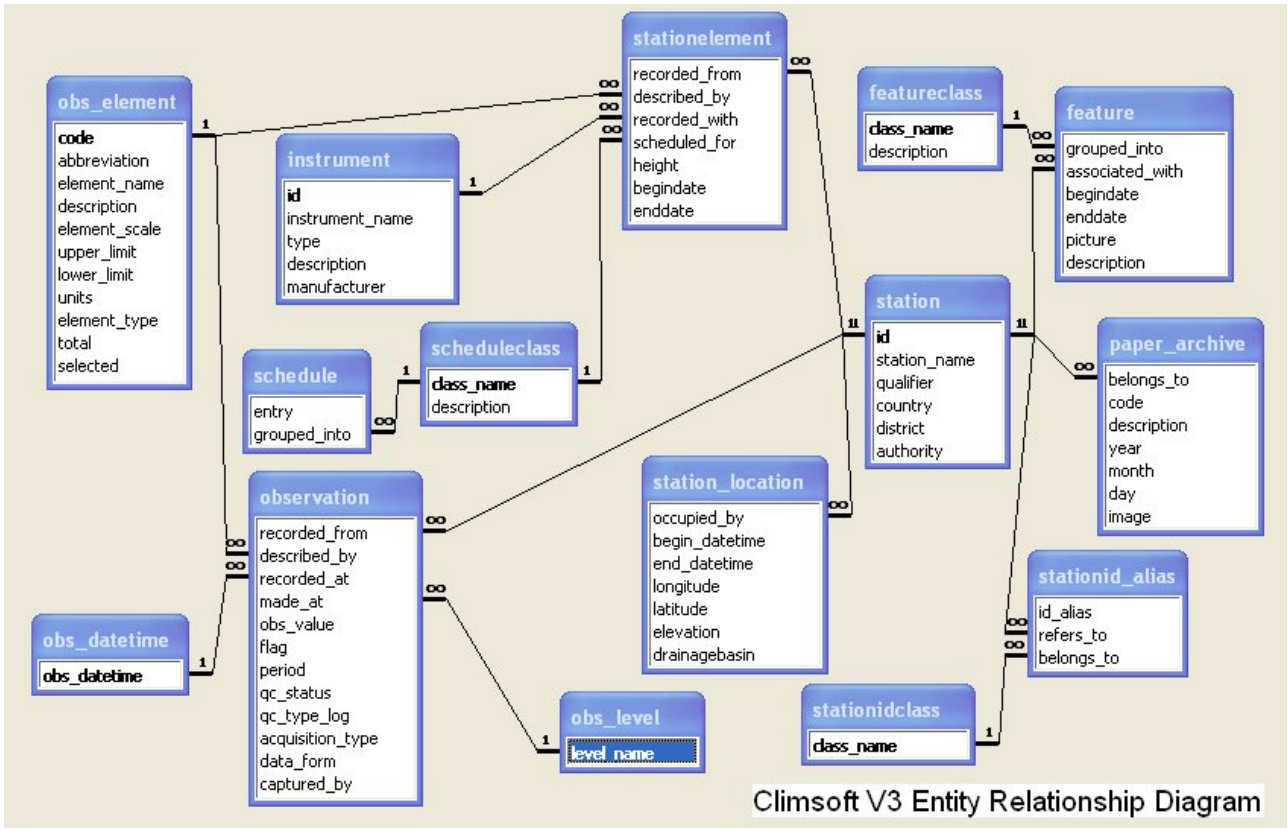
The key features of Climsoft version 3 are as follows:

- No autonumbers for primary keys. This new design improves robustness, efficiency and portability (*New!*)
- No use of SQL reserved words in all database entities and their attributes. Examples of reserved words used in previous versions are *name, element, datetime, scale, level.* (*New!*)
- Increased flexibility to allow capture of data from mobile stations : ships, drifting buoys, aircraft (*New!*)
- Observations stored in one table. Observation values in one column and flags in another column, same record with values (*New!*). Manipulation of both values and flags becomes a lot easier. *The generalized structure of the observation table allows storage of any type of meteorological (or hydrological) data without the need to modify the database design.*
- Capture of station history (*New!*)
- Extensibility to define new elements or new attributes for entities for local use without fundamental changes to data model
- Instrument height in station element (*New!*) Required for TDCF
- Date and time of observation stored as one field of data type datetime (*New!*). This allows easy use of date and time functions.
- Capture of paper archive images for data rescue and also for QC reference (*New!*)
- Additional attribute for observation table to show source of data (acquisition_type) e.g. data from key-entry, imported from CLICOM, ingested from AWS. (*New!*)
- Attribute to show level of QC at record level (qc_status) (*New!*)
- Attribute to show log of QC operations at record level (*New!*)
- Attribute to show user who entered each record of data in observation table (*New!*)
- Attribute to show key-entry form used for entering data into the observation table at record level (*New!*)

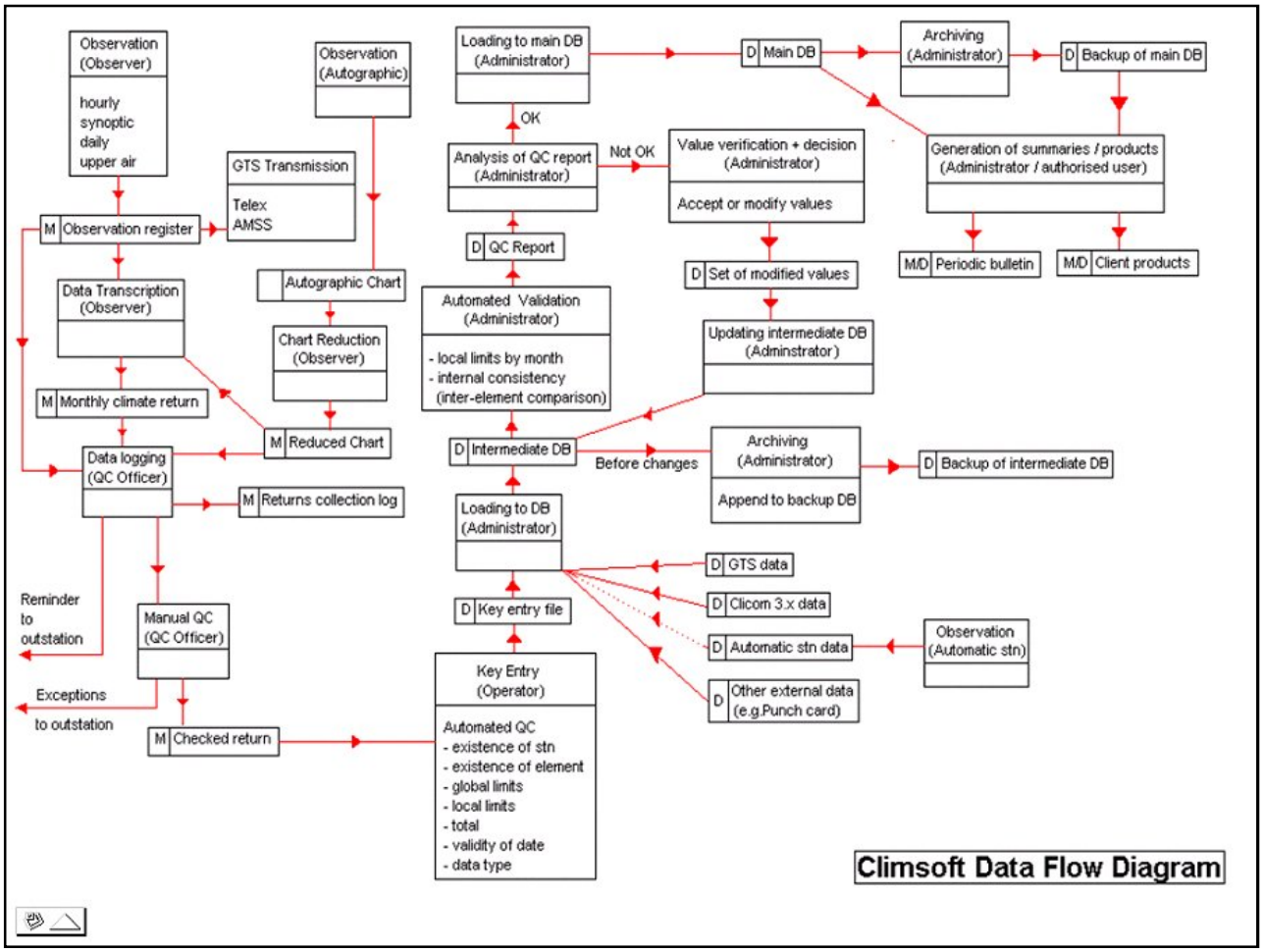
A tabular summary of the database entities is given below.

Table Name	Brief Description
Observation (*Modified!*)	Stores observation values and related information like where the observation was recorded, what the observation is, time of observation etc
Station (*Modified!*)	Station information like name of station, unique primary identifier for a station, country to which the station belongs
Obs_element (*Modified!*)	Information about each meteorological element which could be recorded at a meteorological station
Obs_datetime (*Modified!*)	Date and time when each observation was recorded
Obs_level	Level at which an observation was made e.g. surface or a standard pressure level like 850
Station_location (*New!*)	Meant to capture information about station history, i.e. change of geographical location with time plus the opening and closing of stations. From an object-oriented approach, a land station is considered similar to a mobile station like a ship, drifting buoy or aircraft. Each type of station (observing platform) is considered to have properties like unique identifier, and methods like movement. Apart from storing different coordinates occupied by a station this entity allows storage of multiple entries for opening and closing of a station.
Station_element (*Modified!*)	Information about the measurement of a particular element at a particular station e.g. when the recording of a particular element started at a station
Feature	Information about physical features associated with a station e.g. a big tree near the station
Feature_class	Information about types of features which can be associated with a station
Stationid_alias (*New!*)	Secondary identifier for a station e.g. specific WMO id for a station, ICAO id
Stationid_class	Stores different types of station identifier e.g. WMO id as a group or class of identifiers
Schedule	A set of observation times e.g. 0600, 0900, 1200, 1500
scheduleclass	Name assigned to an observation schedule e.g. AWS, synoptic 1 (0600 to 1500, synoptic 2 (24 hour obs schedule at 3 hourly interval)
Instrument (*Modified!*)	Information about instruments used for observations, e.g. instrument name, manufacturer
Paper_archive (*New!*)	Stores information about images of paper records e.g. code for type of monthly climatological return

ClimSoft Entity Relationship Diagram



Data Flow Diagram



Structure of the Climsoft Program

The ClimSoft program has got two major parts. There is the **user interface** which the programmers and administrators in a national meteorological service may customize to suit their local needs in terms of appearance and functionality e.g. custom data entry forms and products using Visual Basic and Access. Then there is the **Climsoft function library** which can be considered as the core or kernel of the program. An extension of this kernel is the **Logbook**. To all intents and purposes, all what a programmer or administrator would need to know is how to call appropriate functions from the Climsoft library and Logbook library, much like the calling of functions from Visual Basic itself, Excel, Access or the Windows Operating System. There is currently one example where Climsoft is linked to a totally external utility, the freeware **WRPlot** which is used for windrose plotting.

The user interface is made up of modules, local functions, forms and the resource file.

The main module contains code for initializing the ClimSoft program.

Implementing new Products

Introduction: Products in Climsoft are utilities which extract information from the climatic database and display it in some useful way. Climsoft provides some built-in products which perform data extraction, inventory and graphical features (see products in the Climsoft User Guide for details).

If you wish to provide further products to perform specific tasks that are required by your organisation, you can:

- Build new products into Climsoft by adding Visual Basic forms and code into the Climsoft user interface program.
- Build free-standing programs which access the Climsoft database using Data Access Objects (DAO) or ActiveX Data Objects (ADO).
- Build a product as an Excel workbook, which accesses the database using the Microsoft Query facilities built into Excel, and processes them using code (macros) written in Visual Basic for Applications (VBA).

The rest of this section describes how to build a new product into the Climsoft user interface program. Information about DAO and ADO can be found in the Microsoft Access documentation and help files, and information about Microsoft query can be found in Excel documentation and help. A particularly useful and readable reference book on these topics is “Access Database Design & Programming” by Steven Roman (1999), published by O’Reilly, ISBN 1-56592-626-9.

Incorporating a new product into Climsoft: To incorporate a new product into the Climsoft user interface you will need to:

- Create a new form which enables the user to provide parameter values and other information to run the product. The code underlying this form also performs the required actions of the product.

Add a menu item and command button to invoke the new product form, and provide code in existing Climsoft modules to handle these.

Creating a form to run the product: To create the new form:

Use **Project**, **Add Form** to create a new blank form. Give it a sensible name (e.g. frm_MyProduct) and a descriptive caption.

We recommend that all product forms should contain OK, Cancel and Help buttons. These could be copied from another form or created using the toolbox. Give these controls sensible names (the example code below assumes that they are called cmdOK, cmdHelp and cmdCancel).

Add suitable controls to the form to allow the user to specify parameters and other information for running the product. Where appropriate, these can be copied from other forms, and renamed suitably. Some common types of control that may be useful are:

Text Boxes, for specifying text strings (e.g. names) or numeric values.

Combo or List Boxes, that allow the user to choose from a list of pre-defined values.

DataCombo Boxes, that are connected to table columns in the database. These can be used, for example, to list available stations of element types. Note that each Datcombo box needs to have an associated Adodc control (usually hidden) which is connect to the appropriate database table. For this reason we recommend that these controls should be copied from an existing form (e.g. frm_daily_data).

Check Boxes, for indicating whether an optional action is to be done.

Option Buttons, for choosing just one option from a number of alternatives.

Labels, which simply display text for the user's information. Labels have no effect on the action of the form.

Frames, which are used to contain and group together a number of related controls. Frames have no effect on the action of the form, but can help to make the form easier to understand and use

Other types of control can be used as appropriate.

Having put the controls on the form, you now have to write code to make it perform the actions of the product. An outline example of the sort of code that is needed is given under the topic Sample Source Code in the Appendices. The main points to note are:

- **Form_Load**: Called when the form is loaded; connects the **Adodc** control **lookup_station:** with the main database. Also calls LoadResStrings to translate the texts on the form.
- **MyTextBox_LostFocus**: Called when the cursor is moved away from this text box (e.g. by clicking somewhere else). Can be used for example to check the validity of the contents of the box.
- **lookup_station_Error**: An error or warning has occurred in the connection between the Adodc control and the database. These errors seem to be spurious, and so the routine suppresses any error message by cancelling the error.
- **help_Click**: Called when the Help button is clicked. The routine display_help is called to display the help page for this form (Me). The context number of this page is held in the HelpIdContext property of the form.

- **cancel_Click**: Called when the Cancel button is clicked. Simply unloads the form without performing any action.
- **ok_Click**: Called when the OK button is clicked. The code will depend on the required action of the product. For most standard products the general structure is:

    Display an hourglass cursor while data are being extracted.

    Call the library routine **read_registry** to find the Main Database.

    Call the library routine **output_data2** to extract data from the database using the specified SQL strings. If the BaseQueryName is the same as the OutputQueryName, then the BaseSQL does not have to be specified. The routine puts the extracted data into a table with the fixed name **data_products**.

    Call **generate_excel_output** to copy the data from the data_products table into an Excel workbook with the fixed name **excel_output.csv**.

    Unload the form.

Adding a menu item and command button The form frm_products contains a menu item and command button for each product provided in the standard Climsoft system. To be able to invoke your new product using this form, you will need to carry out the following changes to frm_products :

Display the form, and add a new command button and a label describing it (using the Toolbox). We recommend that the name of the command button control should be the name of the product preceded by "cmd_" (for example cmd_MyProduct). You may need to rearrange the existing command buttons on the form.

Use the Menu Editor to insert a new sub-item under the "Products" menu item. If your site uses multiple languages, you may wish to prefix the menu item name with a number which refers to the appropriate string in the Resource File (see Section 4). We recommend that you define the menu routine name (in the box labelled "Name") to be in the form "mnu_MyProduct".

Display the form again, and double–click on the new command button to create an empty subroutine in the code named cmd_MyProduct_Click. This subroutine is obeyed when the user clicks on the button. Insert a line of code to display the new form, thus:

```
Private Sub cmd_MyProduct_Click()
frm_MyProduct.Show
End Sub
```

Click the drop-down arrow in the box in the top left of the code window, and select the new menu item name (e.g. mnu_MyProduct). This also creates an empty subroutine, and the same line of code should be put into it, thus:

```
Private Sub mnu_MyProduct_Click()
frm_MyProduct.Show
End Sub
```

Creating new data entry forms

To add a new data entry form, we strongly recommend that you start by copying an existing form and its associated data structures, and then modify the copy to your requirements as described in Section 2.4. Where possible, it makes sense to choose an existing form which is similar to the one you wish to create. This will reduce the effort needed to produce the new form, and should also reduce the risk of getting things badly wrong.

Unfortunately, you cannot create a new form simply by copying the files of an existing VB form (.frm and .frx), renaming them, and then add the new form files to the program. This is because VB does not allow two forms in the same program to have the same name, and you cannot easily change the name until the new form is added to the program!

To get round this problem, perform the following steps in Visual Basic:

- Use the menu item **Project** **Add Form** to create a new empty form. Change its name and caption to something more meaningful.
- In the **Project** window, select the existing data entry form to be copied by double clicking its name.
- Use **Edit** **Select All** (or **Ctrl+A**) to select all the controls on the form, and then **Edit** **Copy** (**Ctrl+C**) to copy them to the clipboard.
- Select the new form again, and use **Edit** **Paste** (**Ctrl+V**) to insert the controls into it.
- Select the code window of the form to be copied, and select and copy all of the code to the clipboard..
- Finally, select the new form again, and paste the copied code into the new form.

You will now see that both the original form and the new form belong to your project, but are otherwise identical.

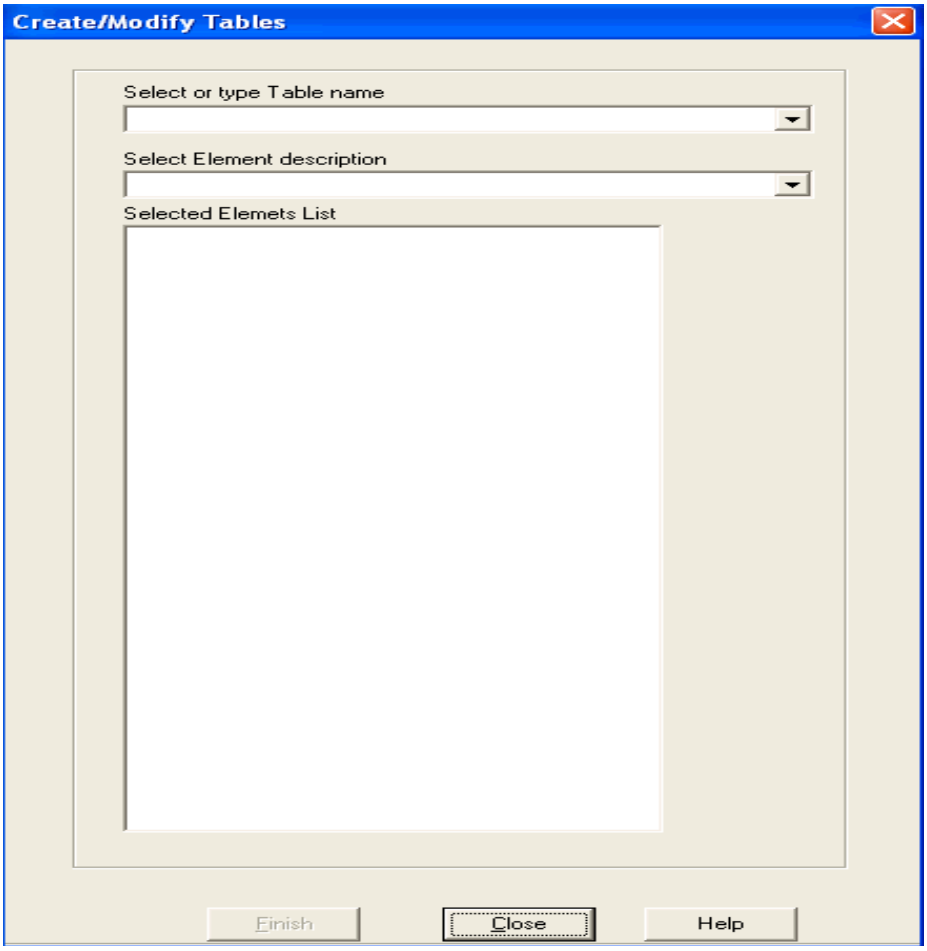
Modifying a Key-entry Form

When modifying a new form, we recommend that the general layout of the form should be kept consistent with existing forms, since users tend to get confused and irritated if forms are inconsistent in style and layout. We also recommend that great care should be taken to ensure that any changes to the properties of the controls and the underlying code are done without changing the fundamental actions of the data entry form. In particular, you should not change the calls to the Climsoft routines which access the temporary work file and the main database.

Modifying Key Entry Forms

From the Main Menu follow the procedures listed below;

1. Click **Tools ➔ Modify Forms** and obtain the following dialog



2. Click on the first drop down list and select form to be modified. If say the form **Synoptic data for many elements for one observation time** is selected all the elements in that form will be listed

3. To add an element on this list click on the element to be proceeded by the added element. Click on the second drop down list and select the desired element from the list. Click on press **Enter** key on the keyboard. The new element is then inserted at the selected position.

4. To remove an element on the list, select it by clicking on it. The press **Delete** key on the keyboard and it gets removed.

5. Step 3 and 4 can be repeated as many times as it is necessary

6. To save the changes click **Finish** and **Cancel** to abandon changes made. **Close** button closes the form.

7. On successful completion of this operation a message indicating the location of the modified table is displayed.

8. Note that modifying a key entry form results in restructuring of its underlying table hence the contained therein will be lost. Should it be necessary to preserve that data it is advisable to make backup and restore it after the form modification

Cosmetic Changes

The following types of changes can be applied to the data entry forms without affecting the action of the form:

Re-arranging the position of the controls on the form.

- Resizing the controls.
- Changing the colour or other aspects of the control's appearance.
- Adding text labels, pictures, frames, tooltips.
- Translating or otherwise changing the text on the form, including the form's caption.

Note, however, that changing the appearance of the form can affect the way that users perceive it, and can lead to confusion, errors or frustration if badly done. Forms should be easy and logical to use, and where possible their use should be self-explanatory.

Adding New Controls

The easiest way to add new controls (for example, new fields in the header or new data boxes in the data entry area) is to copy an existing similar control. This could be a control already on the new form, which you may wish to duplicate in order to add more boxes of the same type. Alternatively, you could copy a suitable control from another form (e.g. to add a new box into the header). In general, it is useful and efficient to treat multiple boxes which have the same function (e.g. values of the same element for each day in the month) as control arrays. In a control array, all controls have the same name but a distinct Index property. They share common event routines, so that the code to support all the controls in the array only needs to be written once. This is both less work and reduces inconsistencies.

If the existing control is already in a control array, VB will automatically add the new control to the array and give it a distinct index value. If the original control is not part of a control array, VB will ask you whether you want to create a control array when you copy the control. It is often sensible to reply Yes. If you reply No, VB will generate a unique name for the new control (which you should then change to a name which indicates the purpose of the control).

Note that most of the controls on a data entry form that allow the user to type text (Text Boxes, Combo Boxes etc) are connected to a field in the Temporary Work File via the Adodc control datPrimaryRS (at the bottom of the form). When adding a control, you need to make sure that its DataSource property refers to datPrimaryRS, and that its DataField property contains the name of a field in the database table corresponding to this form (see Section ???). Each control should refer to a different field.

Removing Controls

To remove a control:

- Delete the control from the form.
- Remove any code that underlies the control's events (unless it was part of a control array), and any other reference to that control in the code

Performing Actions when Events Occur

VB transfers control to an event handling routine whenever something significant happens to the form or the controls on it. These routines enable you to take special actions when an event occurs. If an event handling routine is not included in the code, then no action is taken when the event occurs.

The standard data entry forms contain the following event handling routines:

Form_Load	Sets a local variable to refer to the Climsoft library.
Form_QueryUnload	Prevents the form from unloading. Calls the library routine **close_form** to close the sequencing recordset (if any) and to hide the form.
Form_Keypress	Prevents the form from unloading. Calls the library routine **close_form** to close the sequencing recordset (if any) and to hide the form
Lookup_element_Error	Prevents errors being displayed when the element table is being accessed.
Lookup_station_Error	Prevents errors being displayed when the station table is being accessed
cmdAdd_Click	Called when the AddNew button is pressed. Calls the library routine **addnew** to add a new record and to get the scheduler to insert the appropriate header information.
cmdDelete_Click	Called when the AddNew button is pressed. Calls the library routine **addnew** to add a new record and to get the scheduler to insert the appropriate header information.
view_Click	Called when the View button is pressed. Views the record input so far in an Excel worksheet.
myhelp_Click	Called when the Help button is pressed. Displays the help page for this form.
cmdClose_Click	Called when the Cancel button is pressed. Calls the library routine **close_form** to hide the form. Note that the form should not be unloaded, as it then cannot be displayed again.

Ok_Click	Called when the OK button is pressed. Calls the library routine **ok** to save the record into the temporary work file, and to hide the form.
Reset_Click	Called when the Reset button is pressed. Calls the library routine **Reset** to clear the data boxes in the form.
datPrimaryRS_Error	Ignores warning messages from the connection with the temporary work file.
datPrimaryRS_ WillChangeRecord	Called when the user tries to move to another record. Calls the library routine **validate_data** to perform validation on the whole record.
datPrimaryRS_ MoveComplete	Called when another record is to be displayed. Indicates the record number in the recordset box.

You can add other event handling routines as necessary. Generally, Text and Combo Boxes that you add to the form will not need event handling routines, but you may want to provide one to do extra validation or to take some other special action. You can also insert code into the existing routines, but be careful that these actions do not conflict with the standard actions.

Modifying the User Interface to incorporate a new form

A number of changes need to be made in order to incorporate a new from into the Climsoft User Interface.

Loading the new form

To ensure that the Data Entry program loads the new form when it starts up, and consequently displays its description in the Data Forms box, you will need to modify the *mainmodule.bas* module. In the subroutine Main, add the following line after the existing Load statements:

Load *newformname*

Adding the new form to the list of available forms

The list of available forms that is displayed by the Key Entry form is taken from a table **data_forms** in the Temporary Work File. This table contains a record for each data entry form, with fields:

id	A unique number identifying the record.
order	The order in which the forms are displayed in the Key Entry form. The data entry form are displayed in increasing values of **order**.
table_name	The name of the table in the Temporary Work File corresponding to the data entry form.
form_name	The name of the data entry form.
description	The text description of the form which will appear in the list of data entry forms in the Key Entry form.
selected	Only forms that have this tick box ticked will be displayed in the Key Entry form. This enables you to omit a form from the list without removing it from the **data_forms** table.

You will need to add a new record to this table for your new form. You may also wish to delete (or deselect) records for forms that you do not intend to use.

Making new Data Entry Tables in the Temporary Work File

The Temporary Work File needs to be modified to add a table (and possibly a query) corresponding to the new form. The easiest way is to copy the table and query corresponding to the original form that was used to create the new form.

To copy a table:

- In Access, open the Temporary Work File and display the list of tables.
- Right click the table name for the original form (its name will be *form_formname*), and select Copy.
- Right click on a blank area in the list of tables and select Paste.
- A small dialog will appear. Specify the new table name to be *form_newformname*. Select the option Structure Only, and click OK.

This creates a new table with the same fields as the original table, but containing no data.

If there is a query associated with the original table, display the list of queries and follow a similar procedure to that described above for the table. The new query should be named *qry_form_newformname*.

If you have added a data control (e.g. a Text Box) to the form, you will need to associate it with a field in the temporary work file.

- Open the temporary work file in Access, select the new form table and display it.
- Insert a new field and give it a suitable name.
- You may also need to insert this field in the corresponding query.
- In VB, display the form and select the new control.
- In the Properties window, ensure that the DataSource property contains datPrimaryRS (the name of the database control at the bottom of the form).
- Set the Datafield property to the name of the associated field in the form table.

Similarly, you should remove fields that correspond to any controls that you have deleted.

There may be need to change the behavior of the sequencer.

Mapping a new Data Entry Table to the Intermediate Database

Tables in the Temporary Work File that correspond to a Data Entry form contain one record for each screen of information that is input using the form. So, for example, filling in a screen of data in form_Daily2 (“data for one element for the whole month”) results in a single record containing up to 31 daily measurements (values, flags and periods) for a single element at a station in a given month. Similarly, form_Daily1 (“Data for some elements for one day”) produces a record containing measurements of 7 elements at a station on a particular day.

However, the Intermediate Database (and the Main Database) holds the data in a different structure. Here the basic unit of data is the “observation”, a single measurement (value, flag or period) of an element at a station on a particular date or time. So a single record in the Temporary Work File corresponds to many records in the Intermediate Database. We therefore need to specify how a record in the Temporary Work File is to be mapped onto multiple records in the Intermediate Database.

This mapping is done by MS Access append queries. There is an append query for each data entry form. The design of each append query is unique, depending on the structure of the associated key-entry table.

Structure of the Temporary Work File

The temporary work file is a small Access database which holds the data that have been input using the data entry forms. Once these data have been checked for accuracy and all possible errors have been resolved satisfactorily, they are transferred to the main database for permanent storage. Normally the data in the temporary work file will then be removed, allowing further data to be input into an empty work file.

The temporary work file contains:

- a table (and possibly a query) corresponding to each data entry form. The table has the same name as the form, and holds the raw data that have been input using the form. The optional query, which has the name “qry_” followed by the table name, is used to modify the raw data before they are transferred to the main database, for example to multiply the values by a scaling factor.
- tables of station and element information.
- queries for extracting or modifying station and element information.
- tables and queries for use by the sequencer

No relationships are defined between these tables.

Programming Guidelines

This section gives some guidelines about good programming practices, particularly in Visual Basic. These may help you to produce code which is readable and understandable, and makes errors easier to find and fix.

Naming

- Choose names that make the meaning of the program item obvious. VB allows names up to about 30 characters, so names do not have to be abbreviations.
- It often helps to choose the correct part of speech when naming objects. For example, boolean variable names should describe the “true” state as a word or phrase (e.g. “Finished” or “InputIsValid”). Subroutine names should be commands (e.g. “WriteNextLine”), whereas function names should describe the value returned (e.g. “NextLine” rather than “ReadNextLine”).
- Separate words in long names by an underline character, or to start each word with a capital letter. So WriteDataToOutputFile or write_data_to_output_file are more readable than writedatatooutputfile.
- An advantage of using capitals in names is that you can type them in lower case, and then VB will insert any capital letters that you used when declaring the variable. If VB does not insert the capital letters, this indicates that you have probably made a typing error.
- It can help to prefix a variable with one or more letters to indicate its type and/or scope (for example “i” for and integer, “m” for a variable available to all routines in the module). However, this is a matter of personal taste, as you may feel that this tends to make the name less readable.

Variables

- The scope of variables should be as narrow as possible, so that they can only be referred to by a restricted part of the program. Where possible, declare variables within a subroutine or function, so that they can only be used within that procedure, and other parts of the program cannot affect them. This helps to confine the effects of an error to a small part of the program, so that the problem can easily be traced and fixed.
- If a variable needs to be shared by two or more procedures in a module, then it should be declared at the start of the module using a Dim (or Private) statement. Do not declare it as Public.
- Public variables, which can be accessed by procedures in all modules of the program, should be declared in a separate module, and their meaning and use should be well documented.
- The statement

Option Explicit

should be inserted at the beginning of each module. This forces all variables to be declared before they are used. Otherwise, a misspelled variable name will cause a new variable to be created, which will probably lead to an obscure error when running the program. Note that when Option Explicit is used, VB will not flag an error when you type an undeclared variable name, but will give an error message when you run the program.

- When declaring a variable, always specify its type explicitly, e.g.

```
Dim MyVariable As Integer
```

Otherwise VB assumes that the variable type is Variant, which is much less efficient. Note also that the type should be given explicitly for each variable declared in the same Dim statement; if you declare two variables by:

```
Dim Str1, Str2 As String
```

Str2 will become a String variable, but Str1 will be a Variant! The program will work correctly, but will be less efficient than if you declared each variable as a String, thus:

```
Dim Str1 As String, Str2 As String
```

Code

- Try to modularise your code so that each procedure (subroutine or function) performs a specific task. If you find that the same code occurs repeatedly, this is a sign that the repeated task should be put into a separate procedure.
- Use indentation (with the Tab key) to make your code more readable. We recommend that code should be indented by one step (usually 3 or 4 characters) in the following cases:
    - ☐ In the body of a procedure, between the Sub or Function statement and the corresponding End statement.
    - ☐ Within loops, such as between For and Next statements, or Do and Loop statements.
    - ☐ Code with in If..Else..EndIf structures, in the form:

```
If <condition> Then
<Code if the condition is true>
Else
<Code if the condition is false>
End If
```

    - ☐ Code within Select Case and other similar structures.
- Long lines of code should be broken up into multiple lines using a continuation marker (space followed by underline in VB). This is to make them more readable on the screen and on a print-out. The second and subsequent lines should have a big indentation (at least 2 standard indentations) to make it clear that they continue the previous line and are not a new statement.
- Use comments freely to describe what is happening in the code. We recommend inserting comments in (at least) the following places:
    - ☐ At the start of each module, to describe what it does. It is helpful for the first line to give the name of the module, as this does not have to be specified in any VB statement.
    - ☐ At the start of each procedure (subroutine or function) to describe its purpose, and in the case of a function to describe the returned value.
    - ☐ At the start of any code indentation.
    - ☐ Wherever the action of the code is not obvious.

- ☐ Wherever something especially tricky is being done, such as exploiting a side-effect or using an obscure system function.
- ☐ When code of a previously issued version of the program has been corrected after an error. The comment should include the date of correction and the person doing it, as well as the reason for the correction.

Translating text into other languages

Climsoft uses a **Resource File** to contain text strings that are used by the program, such as menu item names, error messages, form captions, labels on forms, etc. The program refers to these strings by a numerical identifier, which is then converted into a corresponding string contained in the resource file. This allows the strings to be translated into another language without changing the program: only the resource file needs to be modified to contain the translations of the text strings. Indeed the resource file already contains a translation of the strings into French. If your machine language setting is configured to be standard French, the Climsoft program will display the French version of text strings automatically.

The Resource File can be modified using facilities in Visual Basic. In the Project Explorer the Resource File can be found under Related Documents. Double-clicking the resource file name invokes the Resource Editor, which displays the resource file as a folder. This contains another folder String Table, which itself contains two string table resources, the first for the English version, and the second for French. Double-clicking either of these displays the contents of the whole resource file, comprising column of string identifiers (integers) and the corresponding English and French texts. The only difference between double-clicking the two resources is that the language columns are displayed in a different order (i.e. French first if you clicked the French string table).

Having opened the Resource File, its contents can be edited. There are several things that you could do:

- To alter a string in order to correct or replace it, simply click on the string to replace it, or double-click and use normal editing keystrokes to change the text. Do not change the Id column or delete an existing row, as Climsoft will then not be able to display the message held in that row.
- You can add extra strings to the end of the list by clicking in the rightmost existing cell and typing the Tab character. The next Id number in sequence will automatically be inserted in the new row. You can also use the Insert New Row icon.
- You may wish to change the language definition at the top of a column. Typically this would be if your machine is configured for a regional version of French (or English). Double-click the grey heading at the top of the column and use the drop-down list to find the language version that you require.
- You may wish to add translations in a different language. To add a new column, click on the Insert New String Table icon and select the appropriate language in the header. You could then insert the texts in the new language into the column. However, we recommend that you copy the current resource file to the clipboard using the Export Text to Clipboard icon, and paste it into a program such as Word or Excel. The strings can then be translated by an expert without having to use the Climsoft program. When the

translation has been completed, the new text strings can be inserted into the Resource File using copy and paste via the clipboard.

- Note that the first row contains the language identifier used to distinguish the different language versions of the Help files. This is the last two letters of the help file name (e.g. EN for English, FR for French). Help files translated into other languages must have the same name as the existing Help files except for the language identifier.

When you have finished modifying the texts you should exit from the Resource Editor. Note, however, that the Resource File will not be modified until you save it, either explicitly using File,Save Project, or as a result of the save prompt when you exit from the Visual Basic program.

Sequencer

The sequencer handles situations where the contents of a new record's header should follow on logically from the previous record. For example, it could be used when records for a sequence of stations and elements within stations are to be entered, and causes the next station and element in sequence to be inserted automatically into a new record. Similarly, the sequencer can be used to set the date or time fields of a new record to the next date/time in sequence.

Conventionally, the name of a sequence table starts with "seq_", though this is only a convenience for distinguishing sequence tables from other tables. Each record in the sequence table defines the values of the appropriate variables at one point in the sequence. The action of the sequencer is to find the next record in the sequence table. If the end of the table is reached, it starts again at the beginning. Thus the sequence table defines a loop of values to be used in sequence.

The **sequencer** control control is included on all key-entry forms which come with the ClimSoft setup. The property **recordset_name** should be set to the name of the recordset in the **Temp Work File** containing the required sequence of values in the header area of a data entry form. Configuration of the sequencer is first done in the **Temp Work File** before it can be included on a particular data entry form.

Help Authoring

Overview: One way to add help to Climsoft is to make use of “compiled HTML Help”. The contents of the help file can be prepared using a simple text editor (e.g. Notepad) or any other program suitable for preparing HTML files, such as FrontPage.

Each help page or topic is written as a separate HTML file. The set of topic files are then “compiled” into a single file using the Microsoft Help Workshop (freely available as a download from the Microsoft website). This file has the extension “.chm” and is linked to the Climsoft VB project from within VB, via the “Project Properties”.

A freeware help authoring tool called **HelpMaker** provides an easier way to develop help files. HelpMaker is an intuitive graphical utility where help chapters and topics in the development mode appear the same as in the compiled help file. The editing tools to a large extent, resemble those of Microsoft Word. As a result, developing help files using HelpMaker is a lot faster than using most other help authoring alternatives.

The green shading indicated in the HelpMaker screen-shot above indicates the status that topics are still under construction. The **About** box for **HelpMaker** is shown below.

Compilation: At the compilation stage, each help topic is allocated a unique “topic number” that can subsequently be called from within the VB program to display the help appropriate to the context. The allocation of each topic number (called a “context-id” in VB) to a help topic is done by means of a text file that maps the topic numbers to the corresponding html filenames.

Linking Help Files to VB: From within VB, the context-id of a form is set to the value of the relevant topic number in the help file. When a user presses [F1] or clicks on a help button on a form, this triggers a call to a GetHelp (or GetHTMLhelp) function which (all being well) loads and displays the help topic for the form.

Structure of Help Files: HTML files are simply text files containing information that can be displayed using “web browser” software such as Internet Explorer, Opera or Mozilla. Links to other files, headings, paragraphs, lists, images etc. are identified in the text using “tags” to “mark-up” the text and add clarity to the information. The tags are themselves simply text strings enclosed in angle brackets.

Using MS Word: The topic files in a help project can be prepared by using Microsoft Word and saving the file as html, but this puts in a lot of embedded formatting and other codes that cause the resulting help file to be unnecessarily large. To an extent this can be reduced by using the Office “Compact HTML Filter” (check the Microsoft website for details), but in general Word is not ideal for editing html files.

Alternatives: a simple text editor can be used to prepare the html files. Notepad is fine, as is the editor that comes as part of the HTML Help Workshop. However, these do require that you

have a good understanding of html. There are some excellent web-based resources for learning the necessary skills: two good starting points can be found at <http://www.htmldog.com/> and <http://www.w3.org/MarkUp/Guide/>.

Components of a help File: A “help project” is made up of a set of html files, together with any linked images (jpg, bmp or gif files), plus five files with a purpose specific to the project. It is sensible to give these the same name as the project. The extension identifies the purpose, e.g.

climsoft.**hhp** the project file, containing the main project settings

climsoft.**hhc** the contents file, which holds the structure of the help contents window

climsoft.**hhk** the index file

climsoft.**css** the stylesheet file, which holds the html formatting details

climsoft.**h** the map file, which contains a look-up table that associates the help topic files with their corresponding context-id numbers

Building a new project from scratch can be tedious, so the framework of a help system for Climsoft has been prepared with the necessary files and folder structure. Additional files will need to be added for each new dialogue or form that is added to Climsoft.