

WATER RESOURCES OF SOUTH AFRICA, 2012 STUDY (WR2012)

Volume 9: WRSM/Pitman Programmer's Code Manual

AK Bailey & WV Pitman



TT 691/16



WATER RESOURCES OF SOUTH AFRICA, 2012 STUDY (WR2012)

WRSM/Pitman Programmer's Code Manual

Report to the
Water Research Commission

by

AK Bailey and WV Pitman
Original by JP Kakebeeke
Royal HaskoningDHV (Pty) Ltd



WRC Report No. TT 691/16

August 2016

Obtainable from

Water Research Commission
Private Bag X03
GEZINA, 0031

orders@wrc.org.za or download from www.wrc.org.za

The publication of this report emanates from a project entitled Water Resources of South Africa, 2012 (WR2012) (WRC Project No. K5/2143/1) and other projects for the Water Research Commission, the Department of Water and Sanitation and for the University of the Witwatersrand.

This report forms part of a series of nine reports. The reports are:

1. WR2012 Executive Summary (WRC Report No. TT 683/16)
2. WR2012 User Guide (WRC Report No. TT 684/16)
3. WR2012 Book of Maps (WRC Report No. TT 685/16)
4. WR2012 Calibration Accuracy (WRC Report No. TT 686/16)
5. WR2012 SAMI Groundwater module: Verification Studies, Default Parameters and Calibration Guide (WRC Report No. TT 687/16)
6. WR2012 SALMOD: Salinity Modelling of the Upper Vaal, Middle Vaal and Lower Vaal sub-Water Management Areas (new Vaal Water Management Area) (WRC Report No TT 688/16)
7. WRSM/Pitman User Manual (WRC Report No TT 689/16)
8. WRSM/Pitman Theory Manual and (WRC Report No TT 690/16)
9. **WRSM/Pitman Programmer's Code Manual (WRC Report No. TT 691/16 – This report)**

ISBN 978-1-4312-0849-4

Printed in the Republic of South Africa

© WATER RESEARCH COMMISSION

ACKNOWLEDGEMENTS

The authors would like to acknowledge:

The Water Research Commission for their commissioning and funding of this entire project.

The Department of Water and Sanitation for their rainfall, streamflow, Reservoir Record and water quality data, some GIS maps and their participation on the Reference Group.

The South African Weather Services (SAWS) for their rainfall data.

The following firms and their staff who provided major input:

- Royal HaskoningDHV (Pty) Ltd: Mr Allan Bailey, Dr Marieke de Groen, Miss Kerry Grimmer (now WSP Group), Mr Sipho Dingiso, Miss Saieshni Thantony, Miss Sarah Collinge, Mr Niell du Plooy and consultant Dr Bill Pitman (all aspects of the study);
- SRK Consulting (SA) (Pty) Ltd: Ms Ansu Louw, Miss Joyce Mathole and Ms Janet Fowler (Land use and GIS maps);
- Umfula Wempilo Consulting cc: Dr Chris Herold (water quality);
- Alborak: Mr Grant Nyland (model development);
- GTIS: Mr Töbias Goebel (website) and
- WSM: Mr Karim Sami (groundwater).

The following persons who provided input into the coding of the WRSM/Pitman model:

- Dr Bill Pitman;
- Mr Allan Bailey;
- Mr Grant Nyland;
- Mrs Riana Steyn and
- Mr Pieter van Rooyen.

Other involvement as follows:

Many other organizations and individuals provided information and assistance and the contributions were of tremendous value.

REFERENCE GROUP

Reference Group Members	
Mr Wandile Nomqophu (Chairman)	Water Research Commission
Mrs Isa Thompson	Department of Water and Sanitation
Mr Elias Nel	Department of Water and Sanitation (now retired)
Mr Fanus Fourie	Department of Water and Sanitation
Mr Herman Keuris	Department of Water and Sanitation
Dr Nadene Slabbert	Department of Water and Sanitation
Miss Nana Mthethwa	Department of Water and Sanitation
Mr Kwazi Majola	Department of Water and Sanitation
Dr Chris Moseki	Department of Water and Sanitation
Professor Denis Hughes	Rhodes University
Professor Andre Görgens	Aurecon
Mr Anton Sparks	Aurecon
Mr Bennie Haasbroek	Hydrosol
Mr Anton Sparks	Aurecon
Mr Gerald de Jager	AECOM
Mr Stephen Mallory	Water for Africa
Mr Pieter van Rooyen	WRP
Mr Brian Jackson	Inkomati CMA
Dr Evison Kapangaziwiri	CSIR
Dr Jean-Marc Mwenge Kahinda	CSIR
Research Team members	
Mr Allan Bailey (Project Leader)	Royal HaskoningDHV
Dr Marieke de Groen	Royal HaskoningDHV
Dr Bill Pitman	Consultant to Royal HaskoningDHV
Dr Chris Herold	Umfula Wempilo
Mr Karim Sami	WSMLeshika
Ms Ans Louw	SRK
Ms Janet Fowler	SRK
Mr Niell du Plooy	Royal HaskoningDHV

Mr Tobias Gobel	GTIS
Miss Saieshni Thantony	Royal HaskoningDHV
Mr Grant Nyland	Alborak
Miss Sarah Collinge	Royal HaskoningDHV
Miss Kerry Grimmer	WSP
Ms Riana Steyn	Consultant
Miss Joyce Mathole	SRK
Mr Siphon Dingiso	Ex Royal HaskoningDHV

This page was left blank deliberately

CONTENTS

1	INTRODUCTION	1
2	DESIGN PHILOSOPHY OF WRSM/Pitman	3
3	DEFINITIONS	4
4	THE WRSM/Pitman CLASS	6
4.1	General Description.....	6
4.2	How the ObjectEN is determined	8
4.3	Hiding data and functions or methods	9
4.4	Checking and Checking Routines.....	10
4.4.1	Memory Checking	10
4.4.2	Data Checking.....	10
4.4.3	Changed Data	11
4.4.4	Saving object data.....	11
4.5	The Strings.....	12
4.5.1	The Language DLL.....	12
5	FINDING YOUR WAY AROUND THE CODE.....	14
5.1	Variable And Type Definitions: Where To Find Them	14
5.2	Classes: Where to find them.....	15
6	THE GENERAL CLASSES WITHIN WRSM/Pitman.....	16
6.1	The Estring Class	16
6.1.1	The Estring Class: class data record.....	16
6.1.2	The Estring Class: class variables.....	16
6.1.3	The Estring Class: Methods	17
6.2	The Observed data class	19
6.2.1	The Observed data class : Class Data Record.....	19
6.2.2	The Observed data class: Class Variables.....	20
6.2.3	The Observed data class: Methods.....	20
7	THE HYDROLOGICAL MODULE CLASSES WITHIN WRSM/Pitman	22
7.1	The Raingauge Class	22
7.1.1	The Raingauge class: class data record.....	23
7.1.2	The Raingauge class: class variables	24
7.1.3	The Raingauge class: methods	24
7.2	The Observation Point class	26
7.2.1	The Observation Point class: Class Data Record.....	26
7.2.2	The Observation Point class: Class Variables.....	27
7.2.3	The Observation Point class: Methods	27
7.3	The Route Class.....	29
7.3.1	The Route Class: class data record	30
7.3.2	The Route Class: class Variables.....	32
7.3.3	The Route Class: Methods	32
7.4	The Runoff Module Class.....	37
7.4.1	The Runoff Module Class: class data record	37
7.4.2	The SFR Parent/Child parameters	52
7.4.3	The Runoff Module Class: class Variables	54
7.4.4	The Runoff Module Class: Methods	54
7.5	The Irrigation Module Class.....	65
7.5.1	The Irrigation Module Class: class data record	65

7.5.2	The Irrigation Module Class: class Variables.....	72
7.5.3	The Irrigation Module Class: Methods.....	72
7.6	The Reservoir Module Class.....	77
7.6.1	The Reservoir Module Class: class data record.....	77
7.6.2	The Reservoir Module Class: class Variables.....	80
7.6.3	The Reservoir Class: Methods.....	80
7.7	The Channel Module Class.....	84
7.7.1	The Channel Module Class: class data record.....	85
7.7.2	The Channel Module Class: class Variables.....	88
7.7.3	The Channel Module Class: Methods.....	89
8	The classes for Statistics and Plotting.....	94
8.1	The Stats Class.....	94
8.2	The Stats Class: class data record.....	94
8.3	The Stats Class: class Variables.....	96
8.3.1	The Stats Class: Methods.....	96
9	EDITING, COMPILING AND DEBUGGING.....	98
9.1	Resource.F90.....	100
9.2	Resource.RC.....	100
9.3	TiGenLib.....	101
9.4	TiEString.....	101
9.5	WRSM2012.F90.....	101
9.6	WRSM2010.F90.....	101
9.7	WRSM2008.F90.....	101
9.8	WRSM2006.F90.....	101
9.9	WRSM2004.F90.....	101
9.10	WRSM2007.F90.....	103
9.11	WRSM2005.F90.....	103
9.12	WRSM2003.F90.....	103
9.13	WRSM2016.F90.....	104
9.14	WRSM2013.F90.....	104
9.15	WRSM2015.F90.....	104
9.16	WRSM2014.F90.....	104
9.17	WRSM2002.F90.....	104
9.18	WRSM2001.F90.....	104
9.19	.Lib and .mod files.....	104
9.20	Design of Delphi Part of the Code of WRSM/Pitman.....	105
9.20.1	Overview of Delphi-Fortran Connectivity Strategy.....	105
9.20.2	Overview of Database Strategy.....	107
9.20.3	Database Diagram.....	107
9.20.4	Graph Enhancement.....	110
9.20.5	Delphi Source Code Units.....	112
9.21	Final compilation notes following the integration of SPATSIM and database features involving Delphi code, i.e. WRSM/Pitman Compilation Notes.....	114
9.21.1	Development Environments Required.....	114
9.21.2	Compiled Output.....	115
9.21.3	Project Files.....	115
9.21.4	WiDE Settings.....	115
9.22	License.....	121
10	Monthly and daily time step differences.....	122

11	SPECIAL NOTES	124
11.1	Calibration parameters	124
11.2	SAMI Analysis	124
11.3	Screen display – colours	124
11.4	Making fields inactive	125
11.5	Compiler/linker options	125
11.6	Adding rows or columns to screen tables	125
11.7	Adding a new graph - Groundwater	126
11.8	Adding a new graph and adding to a data structure type – Catchment Based Rainfall Massplot	126
11.9	Changing the SSI logo	127
11.10	Adding radio buttons	127
11.11	Adding a tick-box	131
11.12	Transferring certain variables using a GET routine	131
11.13	SPATSIM database settings	131
11.14	Establishing new dialogs	133
11.15	Rainfall files	141
11.16	Establishing a New Menu Item (off the Main Menu)	143
11.17	Establishing a New Grid	143
11.18	Establishing a Pop Up Menu Associated with the Plot Menu	143
11.19	Adjusting length of allowable text in module windows	144
11.20	Changing the Help/About Window	144
11.21	Changing the Groundwater Abstractions in SAMI	145
12	FILE STRUCTURE	147
12.1	The Runoff module parameter file	147
12.2	The channel reach submodel parameter file	154
12.3	Reservoir submodel parameter file	157
12.4	The Irrigation submodel parameter file	158
12.5	The Mine submodel parameter file	176
12.6	Network file example	182
12.7	Rainfile example	185
12.7.1	Raingauge files	185
12.7.2	Rainfile (expressed in terms of percentages of MAP)	185
12.7.3	Report file	185
13	TESTING OF WRSM/Pitman	187
13.1	Testing Procedure	187
14	BUILDING THE WR2005 DATABASE AND CREATING AND RUNNING THE WR2005 INSTALLATION DVD (WR2005 SYSTEM ONLY)	189
14.1	Build WRSM2000 DotNetPaths Program	189
14.2	Run WRSM2000 DotNetPaths Program	189
14.3	Build THE WRSM2000 DatabaseBuilder Program	189
14.4	Run the WRSM2000 DatabaseBuilder Program, i.e. Build the WR2005 Database	189
14.5	Build the WR2005 Quat results (quaternary catchment naturalised flows) program	190
14.6	Run the WR2005 Quat results program (Import quaternary catchment naturalised flows)	190
14.7	Create the WR2005 Installation executable	190
14.8	Run WR2005 Installation DVD (WR2005 system only)	190
14.9	Running the import rainfiles program	190
14.10	Creating the Dashboard	191
14.11	Running the WR2005 dashboard	191

15 WEBSITE (WR2012 study only)	193
15.1 Adding GIS maps	199
15.2 Google Analytics	201
APPENDIX A	203

List of Tables

Table 5-1: Variable, Module and Fortran Source Code	14
Table 5-2: Prefix, Object Type and Fortran Source Code.....	15

List of Figures

Figure 9-1: Database diagram	108
Figure 9-2: Module and Network Tables.....	109
Figure 9-3: Graph logic and Procedures for changes to Fortran code and Delphi code	111
Figure 9-4: Winteracter Development Environment.....	116
Figure 9-5: Libraries	116
Figure 9-6: Compiler Options.....	117
Figure 9-7: Executable	119
Figure 9-8: "Resource.F90" Compilation	120
Figure 9-9: Compilation	120
Figure 9-10: "Resource.rc" Compilation.....	121
Figure 11-1: BDE File Administrator	132
Figure 11-2: Runoff Module Parameters	133
Figure 11-3: Runoff Module: Multiple Calibration.....	134
Figure 11-4: Daily Rainfall File Flowchart of Functions.....	142
Figure 12-1: WQT Type 4: General Screen	165
Figure 12-2: WQT Type 4: Climate Screen	165
Figure 12-3: WQT Type 4: Area Screen	166
Figure 12-4: WQT Type 4: Return Flow Screen	166
Figure 12-5: WQT Type 4: Allocation Screen	167
Figure 12-6: WQT Type 4: Canal Screen	167
Figure 12-7: WQT Type 4: Groundwater Screen.....	168
Figure 12-8: WQT Type 4: Crops 2 Screen	168
Figure 12-9: WQT Type 4: Efficiency.....	169
Figure 12-10: WQT Type 4: Capacity Screen.....	169
Figure 14-1: Diagrammatic representation of folders and datafiles	192
Figure 15-1: Website home page	194
Figure 15-2: Website schematic showing links	195
Figure 15-3: Administration view of the website	197
Figure 15-4: Filezilla screen to upload data.....	198
Figure 15-5: GIS maps folders.....	199
Figure 15-6: Coverages folder.....	200
Figure 15-7: Website Insertion.....	201
Figure 15-8: Google Analytics screen with further options on the right hand side	202

1 INTRODUCTION

This document serves as the design/as built documentation that accompanies the source code of the program WRSM/Pitman as well as the deliverables (DVD in the case of the WR2005 project and the website for the WR2012 project). This document explains the design but it should be seen as supplementary to the in-line documentation within the source code of the program and the two should be read together. The main purpose of this document is not for users but for the current and future personnel managing the WRSM/Pitman code and the website. This document is an update and elaboration on the Programmer's Code Manual that was produced for the WR2005 study and includes all changes made in the WR2012 study.

WRSM/Pitman is a modelling tool to simulate the flow of water through an interlinked network of Catchment, Reservoir, Channel Reach, Irrigation and Mining modules. The modules are linked to one another by means of routes. With the five modules and the routes it is possible to emulate virtually any real world hydrological scenario.

By judicious manipulation of the parameters that steer the way in which runoff is generated and by routing these flows through the various modules, the network can be calibrated against historically observed flows at observation points which are called Gauging Stations.

The nature of the problem that is addressed by WRSM/Pitman is eminently suited to be solved by means of Object Oriented design and programming. Each of the real-world elements (the modules and the routes) within a network can be seen as largely independent entities that can be programmed as independent classes.

For historical reasons, WRSM/Pitman was written in Fortran. The Fortran Compiler Lahey Fortran 90 was used as an update on the earlier versions of Fortran as it is almost 100% compatible and brings the earlier versions of Fortran more in line with modern programming languages. The choice fell on the Lahey compiler largely because it could be used in conjunction with the Winteracter package. The Winteracter package allows one to create Windows style user interfaces with far less effort than it would have taken to do the same by means of either C/C++ or Pascal.

There are certain drawbacks to the Winteracter package, but most of those were easy enough to work around and the benefits of speed and ease of use far outweighed the costs. Those function calls and small functions that were needed to interact with the Windows operating system but were not available in either Lahey Fortran or Winteracter could be written and interfaced without any major problems, mainly because of the excellent interface characteristics of Lahey Fortran.

Additionally, Delphi was used for the database and graphic components.

The source code for Version 2.9 of WRSM/Pitman now spans more than 80 000 lines of code and comments, distributed over 18 Fortran files and 42 much smaller Delphi files. This constitutes roughly seven times more code and comments than its predecessor WRSM90. The reason for this growth is largely the result of the porting from DOS to the Windows operating system, the extra user-interface functionality that is expected from such a program and vastly improved data checking but also the growth in modelling options and functionality.

Although the code features very good in-line documentation, getting to grips with it may, at first, be very daunting unless one understands the basic design of the program, which is described in this document. Once the design is clear, the organisation of the code should fall into place to any programmer with significant experience.

In this document, we will first explain the basic design of WRSM/Pitman and the way object oriented

design and programming was implemented under Fortran. This is followed by a discussion of the functionality of the classes. This section can be used as a reference. Finally, we will run through the program events to illustrate the sequence of calls and messages that follow a particular event.

WRSM/Pitman has been only available in monthly time step form up to 2014. A daily time step version was developed in 1976 but it remained a DOS based version and was not developed into Windowsform with all the user friendly features that the monthly time step has. However, with the daily time step gaining momentum regarding analysis of environmental conditions and operation of dams, it was decided to set up the daily time step code from the DOS model into WRSM/Pitman. This has been done in a different version of the model (version 2.6). The daily time step is also covered in the WRSM/Pitman Theory Manual. Refer to section 10 for details of coding issues for the daily time step version.

2 DESIGN PHILOSOPHY OF WRSM/PITMAN

Because the Fortran 90 language does not have the capability of true Object Orientation, it was decided to implement a form of Simulated Object Orientation.

Simulated Object Orientation is implemented by means of a set of dynamically growing and shrinking arrays of records that contain the data for each instance of an object, and the use of the module / end module language constructs of Lahey Fortran 90.

The Module construct of Fortran 90 allows the definition of distinct program units within the program code. A Fortran module can contain both data and functions (or methods). Within such a module program unit, Fortran 90 allows us to define the data and functions (or, in Object Oriented terminology: methods) as either public or private. By organising the code and the data in modules in this fashion, the modules are, in effect, classes.

The Simulated Object Oriented programming model that we designed allows most of the operations of true object orientation such as overloading of operators and functions and the combination of classes to create composite classes, but it lacks a way to implement inheritance.

Although some purists maintain that inheritance is a fundamental and essential part of Object Orientation, the author does not share this sentiment. In the fifteen years that he has written and repaired Object Oriented code, the author has seen more cardinal programming sins committed with inheritance than with any other Object Oriented Programming feature. The Microsoft Foundation Classes (MFC) in C/C++ are a case in point. Here, the programmers seem to have gone out of their way to prove that inheritance (and more specifically: multiple inheritance) also works ad absurdum. The result is a thicket of interdependent classes where only some of the lower level classes can be used in a stand-alone manner. Any higher level classes require the incorporation of virtually all of the lower level classes, whether they appear directly relevant or not. Modifying the classes themselves is impossible, because even the unmodified source code does not compile properly. As such, deriving stand-alone classes from the existing MFC classes is out of the question. It is therefore hardly surprising that Microsoft is currently in the process of phasing out their Foundation Classes.

During the development of WRSM/Pitman, the lack of the inheritance feature in our programming model never proved to be a problem. Where necessary, we created composite classes. In our composite classes, we included only those additional classes that were absolutely necessary, and then only in the functions (or methods) where they were required. By this means, we kept these 'helper' classes independent of the classes that use them. This adds to the re-usability value of the individual classes.

3 DEFINITIONS

Object

In this document, we will use the term object to mean a specific physical thing. As such, a line would be an object, as would be a runoff area, a channel reach, a route, a raingauge, an irrigation block and so on. In the same way, a program object is a collection of code and data to describe such a physical object.

Attribute

An attribute is a characteristic of an object that is used to model the object.

Class

In this document, we will use the term class for the combination of code methods (functions) and an Object Data Record that we use to model a specific program object.

Method

In keeping with the terminology used in Object Orientated Programming, we will use the term method to indicate a function (or a subroutine) that is part of a class.

Message

In keeping with the terminology used in Object Orientated Programming, we will use the term message to indicate an argument that is passed to a method.

In a call to the method `ES_Add(xx, yy)`, both `xx` and `yy` are messages.

Windows Messages <Index "Windows Message" # "Definition"

Whenever a user does anything – moving the mouse, clicking the mouse button, pressing a key etc. – in a Windows Program, a Windows Message is generated by the operating system and placed in the Windows message buffer. A Windows program must retrieve these messages and act on them.

Object Data Records

Our Simulated Object Oriented Programming model uses a set of dynamically growing and shrinking arrays of object data records. Each object type has its own array of data records, and these arrays grow and shrink in response to the user's requirements.

Instance

An instance is a single case of a class, i.e. all of the class code, but only one specific object data record for that class type. Every data record together with the Fortran code for the class is therefore an instance of the class. A new instance of a class is created by assigning a unique number to a (new or reused) object data record.

Encapsulation

Encapsulation is the 'hiding' of data and complexity that is contained in a class from the rest of the

program. When encapsulation is implemented as it is in WRSM/Pitman, it is only possible to write data to a particular instance of a class by means of a method that was specially written to do so. Similarly, data can only be retrieved from a particular instance of a class by asking for it – again by using a method.

Different instances of a particular class within WRSM/Pitman are encapsulated – inherently, they know nothing about one another – and any method in a class only works on one particular instance of that class at any given time.

Overloading

A method that works for two or more different types of messages is called an overloaded method.

As an example: the method ES_Add in the EString class can be called as:

```
ES_Add( string_variable, 0 )
```

or as

```
ES_Add( integer_variable, 0 )
```

or as

```
ES_Add( float_variable, format, 0 )
```

etc.

In a non-object oriented design, this would have required three separate functions and depending on the message (argument) different functions calls would be used (e.g. ES_AddString, ES_AddInteger and ES_AddFloat). Since the method ES_Add is overloaded, however, we need to use only one call (ES_Add) for all the different cases; only the messages that are used are different. By using different messages, the function ES_Add works out by itself what is meant and does whatever is needed.

Parent and Child

These terms were originally coded as “Master” and “Slave” but were found to be undesirable terms and were replaced by “Parent” and “Child”. Function names, variables, etc. in the code have been left with “Master” and “Slave” references (as only a programmer will ever see them), therefore RU_IsParent will actually be RU_IsMaster in the code.

4 THE WRSM/PITMAN CLASS

The general description of a WRSM/Pitman class is best done by means of an example.

Note to those familiar with Object Oriented Design: The example only serves to illustrate our use of the Object Data Record and the structure that was used throughout WRSM/Pitman. It is not meant as a primer for Object Orientated Design. We will not discuss the idea of base classes, inheritance and combined classes at this stage, because that would make the example that much harder to follow. Hence the absence of a Point-class which could have been proposed as a base class.

4.1 General Description

Suppose that we have the object "Line". A line (as on a 2-Dimensional drawing) is defined as having a start point and end point. Each point has two coordinates – for example X1 and Y1 at the start point and X2 and Y2 at the end point.

One possible further attribute of the line is the line-thickness with which the line that connects the two points must be drawn. There could be many more attributes for the line, such as colour, visibility and so on, but for the sake of simplicity we will consider only this single attribute.

In Simulated Object Orientation, the data record for a line object would therefore have to contain the following elements:

Integer X1
Integer Y1
Integer X2
Integer Y2
Integer LineWidth

To distinguish one Object Data Record of the Object type "Line" from the next, we have to add a unique number to each record, just like every Object in a true Object Oriented language has a unique number (a pointer) that is assigned to the Object when it is created.

It is not feasible to use the index number of the record in the array of records for this purpose, because records may be deleted or moved about within the array. The actual position of an object record within the array may therefore change from time to time.

To achieve a unique number, we therefore add another element to the Object Data Record and call it ObjectEN. We could have called this number the ObjectID or the ObjectIN but for historical reasons, we decided on ObjectEN (the object's External Number, since the user has some influence on this number). The word *Object* in *ObjectEN* is a placeholder. In the case of the line object we would use the record variable LineEN for this number.

The Object Data Record will therefore now have the elements:

Integer	LineEN
Integer	X1
Integer	Y1
Integer	X2
Integer	Y2
Integer	LineWidth

It may be that this ObjectEN is not explicit enough to a user's liking. The numbers for the ObjectEN must

be unique and they are assigned sequentially from 1 to 9999 (see: "How the ObjectEN is determined", below). If a specific line object is deleted, the numbering will no longer be sequential and could therefore be a little confusing. For this reason, it was decided that every object should have an ObjectNAM (Object Name) in addition to the ObjectEN. In this way, an Object Data Record may be accessed by two means: by means of the ObjectEN or by means of the ObjectNAM. The third means (accessing a record by means of its position in the array) is also used in some cases, but this is done in private functions (functions that are only visible internally in the class) only.

In Lahey Fortran 90, a record is defined as a user defined data type. It was therefore decided to call each record data type after the type of record that it represents. In the case of the line, the data type would be called Line_Type.

The form of the Line Object data record (or user (i.e. programmer) defined data type Line_Type) is therefore:

Integer	LineEN
Integer	X1
Integer	Y1
Integer	X2
Integer	Y2
Integer	LineWidth
Character(Len=20)	LineNAM

It is possible to create a data record of the type Line_Type dynamically. In this process, the creation of the record returns a pointer to the place in memory where this new record was created. It therefore makes sense to use these pointers to the records, rather than to create an array of records at the start of the program and hope that the size of the array will be large enough to cope with the demands of the user.

Since we are going to use the pointers to the newly created records, we define a pointer variable to store the pointer to a newly created Object Data Record. Generically, this variable is called the Object_PTR. In the case of the line object, this pointer variable would be called Line_PTR.

In addition to Line_PTR, we also need a temporary pointer to an Object Data Record for all sorts of operations. Generically, we call this temporary pointer Temp_Object_PTR, or, in the case of the Line: Temp_Line_PTR.

Since we use the dynamic creation process of records, all that we have to keep track of is an array of pointers to the object data records that have been created. This array of pointers is a dynamic array, which means that it can grow and shrink on demand.

Generically, we called this array of pointers to the Object data records the Object Pointer Array. Generically, we call this pointer array the Object_PA. In the case of a line, the array would be called Line_PA.

The Object_PA (Object Pointer Array) which, in the case of the line is called the Line_PA, grows and shrinks dynamically. To achieve this, we use a dynamically allocated temporary array in which the pointers can be stored for a time, while the actual Object pointer array is deleted and created again with either a larger or a smaller dimension. Once the new array is created, the pointers are copied back to the Object_PA. Generically, we called this temporary array the Temp_Object_PA.

In the case of the line, this array would be called the Temp_Line_PA.

When the program starts, there are no data records for any objects, and therefore the arrays Object_PA and Temp_Object_PA do not yet exist. For the Line object, these variables would be called Line_PA and

Temp_Line_PA, but neither would exist. Once the user specifies that a Line object is to be created, the arrays themselves are created with three elements – Line_PA(1), Line_PA(2) and Line_PA(3).

Every class contains two counters with the generic names: MObject and NObject. Only the first two letters of the object's name are used in the stead of the 'Object' part of the variable name and so in the Line class these variables would be called MNLI and NLI. The variable MNLI contains the maximum number of Object Data Records that are available at this point in time, and the NLI variable indicates how many of these Object Data Records actually contain data.

Every time data is placed into an Object Data Record for a line, the variable NLI increases by 1.

Once all three object data records are filled (i.e. NLI = MNLI), and a further Line object needs to be added, a Temp_Line_PA array is created with 3 extra elements (i.e. Temp_Line_PA(1) to Temp_Line_PA(6)) and the 3 pointers that already exist are copied from the Line_PA array to the Temp_Line_PA. If this is successful, the variable MNLI is set to 6 (but NLI remains at 3). Then the original Line_PA array is deleted and Temp_Line_PA is renamed to Line_PA.

When a Line object is deleted, the actual Object Data Record that is sitting somewhere in memory is deleted. Once this is done, the 'gap' that is left in the Line_PA is closed. All the pointers that occur below the pointer that is now invalid are shifted upwards.

This strategy has the major advantage that if there were other object types in the program (for example the objects Circle, Triangle, etc.) no memory space would be taken up by an array that may or may not be used by the user. Hence the only limiting factors to the number of objects that the user may have in his system would be:

- That there can only be 9999 objects of a specific object type and
- That there is enough memory space available on the computer on which the program runs.

If there is enough memory available, therefore, the user could have (for example) 9999 Line Objects, 9999 Circle Objects and 9999 of each and every other object type that may be defined in the program.

We decided on 9999 because one has to stop somewhere, and so must the numbering, as will be explained below.

4.2 How the ObjectEN is determined

The ObjectEN (Object External Number) which is a part of every Object Data Record is determined by either the data file that is used to import data into the model or by the class of the object itself. Although a given ObjectEN is unique to a given Object Type (i.e. Line, Circle, etc.) it is not unique to a Network as a whole – it is perfectly possible to have, for example, a Runoff Module with the EN number 1, a Channel Reach with the EN number 1 and a Route with the EN number 1.

Where a user instructs the program to import data from a file into the program, the data file will have the ObjectEN in its data. The program will check whether an Object Data Record with the specified ObjectEN already exists. If a data record with the specified ObjectEN does not yet exist, the program will accept the number in the file.

In the event that an Object Data Record of the type specified already exists, the program will automatically assign the next ObjectEN number that is available. To do this, the program uses the method xx_NextNewEN. This method goes through all the Object Data Records of the specific Object type and determines the next possible new ObjectEN. When the next number that it finds is greater than 9999, the routine starts again from 1 and checks whether there are any 'gaps' in the numbering.

Gaps could occur if an object is deleted from a system. If, for example, a Network was loaded and this network contains three Line objects, these objects would have the LineEN variables 1, 2 and 3. Suppose, now, that the line with LineEN of 2 is deleted and a new Line added to the network. The new LineEN that is assigned will be LineEN 4. LineEN 2 is left alone because it could be that the user made a mistake when he deleted it, and he might want to import it again when he spots his error.

If this does not happen (i.e. the line with the LineEN of 2 is not loaded again), then once 9999 LineENs are allocated, the `xx_NextNewEN` will go through the list of records again, spot that LineEN 2 is not assigned and assign the new LineEN as 2.

When the user specifies that a new object is to be added to a Network, the program uses the same function (`xx_NextNewEN`) to determine the EN for the new object. The user can change this number if he so desires. If the number is changed, it is checked to make sure that no duplicate ENs exists for this object type in the Network.

4.3 Hiding data and functions or methods

True to the philosophy of Object Orientation, the data of the Objects is hidden from the rest of the program.

In WRSM/Pitman it is impossible to access any of the Object Data Records directly from outside the Object module. This is done by making use of the module constructs of Fortran 90. By means of this construct, it is possible to define all data – and some of the functions – as private. By explicitly defining some of the functions in the module as Public, it is possible to allow controlled access to the data from other parts of the program.

The data is hidden so effectively that even the function that creates a new Object Data Record is hidden from the rest of the program.

To create a new Object Data Record, the generic function

```
Object_Read(ObjectEN, FileName, ReadInstruction)
```

is used. In keeping with the nomenclature that was used before, the functions in the classes are named according to the first two letters of the name of the object. In the case of the Line object, this function would therefore be called `Li_Read()`.

Depending on the messages (or arguments) that are send to `Object_Read()`, it may use the file name supplied to load data for the object or it may create an empty data record that can be edited later to add the necessary data.

This was done because there are all manner of checks that must be carried out before and after a new record is created to make sure that everything will go (or went) according to plan. Before an object record is created, the function `Object_Read()` checks whether an object record with that ObjectEN already exists. The function fails when this is the case and a new unique ObjectEN is used. Once the creation operation was carried out, the function `Object_Read()` checks whether there was enough memory available to create the record. Again the function fails if this was not the case. The necessity for these checks precludes a direct creation of a new object by means of a call to `Object_New()`, and a further function (for example `Object_NewObject()`) would have added extra complexity or a duplication of code.

4.4 Checking and Checking Routines

4.4.1 Memory Checking

As the program runs, object data records are created and destroyed. When the object data records are created, memory is assigned, and when object data records are destroyed, this memory is released again.

It stands to reason that the program should keep track of the availability of memory on the computer on which it runs or risk a disastrous breakdown. For this, we incorporated the global variable `MemoryOK`. This is a logical (or Boolean) variable that is checked and (if necessary) set by any function that either creates or destroys object data records. All global variables are kept in one place: in the `globals_module` in the file `wrsm2000.f90`.

When the program starts, the variable `MemoryOK` is set to `TRUE`. When the assignment of a block of memory fails, the function that tried to assign the memory will set `MemoryOK` to `FALSE`. Because `MemoryOK` is in the global variable class it is a sticky variable and will remain true or false for all objects in the program once it is set.

If, at any stage, the user releases some of the memory by destroying one or more of the object data records, the variable `MemoryOK` is reset to `TRUE` if it was set to `FALSE` before. This may or may not be helpful because certain object data records are larger than others and a block of memory may only be assigned in a single contiguous block. If a number of small Object Data Records are deleted, therefore, it is not certain that one large Object Data Record could be assigned in their stead. Even if there is theoretically enough memory available in small, fragmented units, therefore, the assignment of a single larger chunk may still fail and `MemoryOK` will once again be set to `FALSE`. There is currently no known way to consolidate fragmented program memory, other than to save all data and closing and restarting the program. The same problem has been encountered in other programming languages, notably in Microsoft's Visual C/C++. It is thought that the Fortran 90 compiler 'inherited' this flaw from Microsoft.

Under no circumstances should a class be modified to contain `MemoryOK` as a class variable or as a local variable in a function. If `MemoryOK` is added to a function, the scope rules dictate that this variable will be used instead of the global one, with all the predictable mayhem that this will cause.

4.4.2 Data Checking

There are a number of data checks that are carried out by the classes on the object data before the data is used. Since wrong data will produce wrong results, an Object Data Record is only deemed valid once it has been checked. In `WRSM/Pitman`, every Object Data Record therefore contains the Logical (or Boolean) variable `Valid`.

Shortly after an Object Data Record is created, the function `Object_Init()` is called. This method sets the object data variable `Valid` in the record to false.

Before the object data is used, a routine called `Object_Check` is called which checks that all the data in the object data record is correct (or at least feasible). If all the tests are positive, the variable `Valid` in the relevant Object Data Record is set to true.

If we go back to our example of the line class, we would therefore add the Boolean variable `Valid` to the Object Data Record. The class method `Li_Init` would set the variable `Valid` in the relevant record to False, and the class function `Li_Check` would set the variable `Valid` to true if the data passed all the tests.

The object data record would therefore now contain the variables:

Integer	LineEN
Integer	X1
Integer	Y1
Integer	X2
Integer	Y2
Integer	LineWidth
Character(Len=20)	LineNAM
Logical	Valid

4.4.3 Changed Data

Of course the user may wish to edit the data in one or more object data records. This, too, is done from within the program. When he asks to edit the data, the program calls a routine with the generic name `Object_Edit()`. In the case of the line class, this function would be called `Li_Edit`.

If the user does, indeed, change some of the data in the specified object data record, the program needs to keep track of this. If the data has changed, it must be saved at some point before the program ends or before a new network is started (or loaded) or the changed data will be lost.

For this purpose, every object data record contains the logical (or Boolean) variable `Changed`. When an Object Data Record is retrieved from a file or a database, the variable `Changed` is set to `FALSE`. As soon as the editing routines detect a change in the data, the object data record variable `Changed` is set to `TRUE`.

When the user tries to terminate a network or close the program, all object data records for all object types are checked to see whether the data was changed. This is done by means of the class function which is called (generically) `Object_HasChanged`. In the case of the Line, this function would be called `Li_HasChanged`. If this function returns `TRUE`, the user is asked whether he wishes to save the data.

Adding this variable to the Line Object data record, we see that the record will now contain the following variables:

Integer	LineEN
Integer	X1
Integer	Y1
Integer	X2
Integer	Y2
Integer	LineWidth
Character(Len=20)	LineNAM
Logical	Valid
Logical	Changed

4.4.4 Saving object data

Of course the program needs to know where to save the data in the object data records. A data record is either loaded from a file, from a database or is created from scratch. When the function `Object_Read()` is called and the messages (or arguments) to this function dictate that the data is to be read from a file, this file name is saved in the object data record variable that is generically called `ObjectFile`. In the case of the line, we use the first 2 letters of the object's name, and so this variable would be called `LiFile` in the case of the Line object. Strictly speaking, this distinction

between the classes was not necessary, but it was kept for historical reasons, since the user has some control over the name of the file.

The object data record for our Line object example would therefore now grow to:

Integer	LineEN
Integer	X1
Integer	Y1
Integer	X2
Integer	Y2
Integer	LineWidth
Character(LEN = 20)	LineNAM
Logical	Valid
Character(LEN=x)	LiFile

When a new object is created from scratch, a standard file path is created for the file in which the data will be saved later. This file path is composed of the input folder that the Network uses, the Network abbreviation, the (abbreviated) name of the object type, its unique ObjectEN number followed by .DAT.

If the input folder for the network was C:\WRSM2000\Data\NorthWest\ and the Network code was NW, then line object number 10 would be assigned the file path C:\WRSM2000\Data\NorthWest\NWL10.DAT.

Once it is created, the filepath is placed in the object data record, from where it is read by the method with the (generic) name Object_Save. To save a line, the method Li_Save would be used.

4.5 The Strings

4.5.1 The Language DLL

The strings that are used in the program to communicate with the user were not placed in the source code itself, but in a DLL that is queried by the program whenever a string is needed. This was done for two reasons:

1. If they are embedded in the program code itself, strings take up a large amount of space in the memory. If strings are embedded in the code, they are always loaded into the memory, whether they are actually used or not, and remain there until the program is terminated. If we use a DLL, the strings remain on the hard drive and are imported individually (or in small blocks) into memory as and when they are needed, thus saving on memory space. When the strings are kept in a DLL, the actual executable file will therefore have a smaller footprint in memory. The smaller the footprint of the program in memory, the more space will be available for the data.
2. South Africa has 11 official languages. In the event that a version of the program is required in a language other than English, this could be implemented relatively easily – it would only involve the translation of the strings within the DLL. If the strings were scattered in amongst the code, this would be much harder to do. In addition, the footprint of the program would be increased – not only because of the large number of strings that would be duplicated, but also because extra code would be needed to select the correct language that was selected by the user.

The DLL contains only one small routine that allows a program to attach to it and a normal resource string lookup table. We have used the Microsoft Visual C/C++ version 5 compiler to create the language DLL, but because it is so simple, any other compiler that can create a DLL could be used to create it. Unfortunately, the Winteracter environment does not allow for the number of strings that we need in our language DLL – otherwise we would have used that compiler.

To retrieve a string from the language DLL, we make use of a C-routine encapsulated in a Fortran function called GetString(string_number). This method resides in the library LGLib. The source code for the library LGLib is not freely distributed for reasons that will be discussed later.

The first time that the method GetString() is called, it automatically attaches the language DLL (currently only "WREng.DLL") to the program.

Once the DLL has been successfully attached, the routine checks whether the DLL has the correct version number. This is done because the program is updated from time to time and so it may happen that the new version of the program is installed, but only an older version of the language DLL is available. The number of the DLL version that is expected is specified in the main program, in the constants_module in the file "WRSM2000.F90". This number is also placed in string number 29 in the resources of the language DLL. If these two numbers are not the same, the user is warned that he is using the wrong version of the Language DLL, and that some of the messages that he may get from the program will either be blank or garbled. After this message, the program will continue, but as warned, some of the messages may not make much sense.

As an alternative to compiling with the Microsoft Visual C/C++ version 5 compiler, Delphi version 7 or later can be used. The procedure using this compiler is as follows :

The "WREng.rc" file is a text file and contains all the strings required in the "WREng.dll" . This file can be edited or added to with any text editor. It consists of blocks of 15 strings which each have a unique number.

Once the "WREng.rc" file has been updated, load the file "WREng.dpr" into Delphi and Build (compile). This build will also make use of the "resource.h" datafile to produce "WREng.dll" and "WREng.res" datafiles.

Move the "WREng.dll" into the same folder as the "WRSM2000.EXE" program.

A new file has been added to interface with the database, namely : "WRSM2000DB.dll"

A further file was added to keep track of version numbers of executables, namely : "version.rc" . Placing the cursor over the WRSM2000.EXE file will show relevant information about the version.

5 FINDING YOUR WAY AROUND THE CODE

5.1 Variable And Type Definitions: Where To Find Them

Table 5-1: Variable, Module and Fortran Source Code

Variable	Module	Fortran source code file
Graph_Type	types_module	WRSM2000.f90
GraphParam_Type	types_module	WRSM2000.f90
INDIRCHANGED	globals_module	WRSM2000.f90
LANGVERSION	constants_module	WRSM2000.f90
MAJORVERSION	constants_module	WRSM2000.f90
MemoryOK	globals_module	WRSM2000.f90
MINORVERSION	constants_module	WRSM2000.f90
MNCRIN	constants_module	WRSM2000.f90
MNCROUT	constants_module	WRSM2000.f90
MNFILC	constants_module	WRSM2000.f90
MNRRAGT	constants_module	WRSM2000.f90
MNRUAL	constants_module	WRSM2000.f90
MNRUFOR	constants_module	WRSM2000.f90
MNRUOUT	constants_module	WRSM2000.f90
MNRUPAV	constants_module	WRSM2000.f90
MNRVAVT	constants_module	WRSM2000.f90
MNRVIN	constants_module	WRSM2000.f90
MNRVOUT	constants_module	WRSM2000.f90
MNTFP	constants_module	WRSM2000.f90
MNTPS	constants_module	WRSM2000.f90
MNYRS	constants_module	WRSM2000.f90
MODSCHECKED	globals_module	WRSM2000.f90
R2_Data	R2KTypes_module	WRSM2000.f90
R2_Type	R2KTypes_module	WRSM2000.f90
ROUTESCHECKED	globals_module	WRSM2000.f90
SUMWRITTEN	globals_module	WRSM2000.f90
SYSRUN	globals_module	WRSM2000.f90
TEXTHCOSSET	globals_module	WRSM2000.f90
WRERRFILEOPEN	globals_module	WRSM2000.f90
YEARSCHANGED	globals_module	WRSM2000.f90
YearValue_Type	types_module	WRSM2000.f90
YearValue2_Type	types_module	WRSM2000.f90

5.2 Classes: Where to find them

As discussed above, the program is divided into Fortran 90 program units called modules. Because some of the methods (or functions) and variables can be defined as public or private, these modules can be regarded as classes. Private methods can only be called from functions within the class whereas public methods can be called from any other module or function that contains the statement “use xxx_module”.

To distinguish the functions within the different modules, but to keep a uniform nomenclature of the methods, each method (or function) name contains a two letter prefix.

So, for example, the program has a number of functions called xx_Edit. You will encounter RU_Edit, RR_Edit, RV_Edit, CR_Edit, etc.. These functions do the same sort of thing on different types of data sets, depending on the prefix.

Table 5-2: Prefix, Object Type and Fortran Source Code

Prefix (xx_)	Object Type	Fortran source code file
CR_	Channel Reach	WRSM2003.f90
ES_	Error String	WRSM2011.f90
GR_	Graph Renderer	WRSM2014.f90
OB_	Observation Point	WRSM2008.f90
OD_	Observed data class	WRSM2008.f90
R2_	Rainfile Creator	WRSM2015.f90
RG_	RainGauge	WRSM2010.f90
RR_	Irrigation block	WRSM2007.f90
RT_	Route	WRSM2006.f90
RU_	Runoff area	WRSM2004.f90
RV_	ReserVoir	WRSM2005.f90
SG_	System Global (Network)	WRSM2012.f90
ST_	Statistics Calculator	WRSM2013.f90
SY_	SYstem (or Network)	WRSM2002.f90

6 THE GENERAL CLASSES WITHIN WRSM/PITMAN

In this section we will discuss the more general (helper) classes that are used within WRSM/Pitman. These classes are discussed first because they are used in the Hydrological Module Classes that are discussed later.

6.1 The Estring Class

6.1.1 The Estring Class: class data record

None – only one instance of the class exists in the program.

6.1.2 The Estring Class: class variables

! The maximum length of the string in which to write the messages (set to 2048 characters)

```
INTEGER, PARAMETER :: ERROR_STRING_LENGTH = 2048
```

! The string that is used everywhere to write the messages.

```
CHARACTER (LEN=ERROR_STRING_LENGTH) :: ErrorString = ''
```

! Whether or not the error file is open.

```
LOGICAL :: WRERRFILEOPEN = .FALSE.
```

! Whether or not the trace file is open (not implemented yet).

```
LOGICAL :: TRACEFILEOPEN = .FALSE.
```

! Whether errors were written to the error file

```
LOGICAL :: ERRSWRITTEN = .FALSE.
```

! Whether program traces were written to the error file (not implemented yet)

```
LOGICAL :: TRACEWRITTEN = .FALSE.
```

! Whether the user specified that all files may be overwritten

```
LOGICAL :: GOVERWRITEALL = .FALSE.
```

! The logical unit at which the error file is opened

```
INTEGER :: KIN1
```

! The logical unit at which the trace file is opened (not implemented yet)

```
INTEGER :: KIN2
```

6.1.3 The Estring Class: Methods

6.1.3.1 *Public methods*

Public methods can be called from any other module or function that contains the statement

```
use estring_module
```

ES_Add

This is an overloaded method. It takes the various message types that conform to the formats of private functions ES_AddString, ES_AddChar, ES_AddInt, ES_AddFInt, ES_AddReal and ES_AddSI and, using these functions, adds the data contained in them to the class variable ErrorString.

ES_Ask

Shows a message box on the screen with the current contents of ErrorString with Yes and No buttons. The button that is highlighted depends on default. To get the answer, use the call WInfoDialog to get either 0 or 1

Note: this subroutine should not be used when a sub-dialog is required since the call to WInfoDialog will return the last button pressed in the Modeless or semimodeless dialog – in such cases, use ES_Query instead.

ES_Clear

Clear the class variable ErrorString.

ES_CloseEF

Close the error file (if it is open)

ES_CloseTF

To close the Trace File (not implemented)

ES_ErrsWritten

Reports whether errors were written to the error file or not. (true or false).

ES_Get

Return the contents of the class variable ErrorString

ES_GetFileUnitEF

Returns the Error File Fortran file unit number

ES_GetOverwriteAll

Report whether the user has specified that the output file(s) that were created by the EString class should all be overwritten

ES_IsErrFileOpen

Report whether the Error file that can be created in the EString class was opened.

ES_IsTraceFileOpen

Report whether the Trace file that can be created in the EString class was opened.

ES_Length

Report the length of the class variable ErrorString, without the trailing blanks.

ES_NullStrip

Strip the NULL (\0) which marks the end of a C-string from the class variable ErrorString

ES_NullTerm

Add a NULL (\0) to the end of the class variable ErrorString to use the string as a null terminated C string..

ES_OpenEF

Open an Error File with the specified name

ES_OpenTF

Open a Trace File with the specified name

ES_OutOfMem

Place a message box on the screen to report that the class has detected a memory error

ES_Query

Show the contents of the class variable ErrorString on the screen with 2 buttons, Yes and No and report whether the user pressed the Yes or the No button as 1 and 0.

ES_SetOverwriteAll

Set the internal variable to indicate that all files (e.g. the Error File and the Trace File) that can be created by the class may be overwritten when a file output query is made

ES_Tell

Show the contents of the class variable ErrorString in a message box with a single OK button.

ES_Write

Write the contents of class variable ErrorString to the file specified as unit number KOUT. The file must have been opened elsewhere. Report any problems.

ES_WriteEF

Write the contents of the class variable ErrorString to the Error File (if the file was opened) .

ES_WriteTF

Write the contents of ErrorString to the trace file (not implemented yet).

6.1.3.2 Private methods

Private methods can only be called from within the class.

ES_AddString

Add a string of any length to the class variable ErrorString (max length check)

ES_AddChar

Add a single character to the class variable ErrorString (max length check)

ES_AddInt

Add an integer to the class variable ErrorString (max length check), uses ES_AddString

ES_AddFInt

Add a formatted integer (e.g. Octal or Hex) to the class variable ErrorString (max length check), uses ES_AddString

ES_AddReal

Add a real value to the class variable ErrorString (max length check), uses ES_AddString

ES_AddSI

Add a short integer (INTEGER(2)::) to the class variable ErrorString (max length check), uses ES_AddString

6.2 The Observed data class

The Observed data class is a single instance class that is used by the Observation Point Class and the Stats class.

6.2.1 The Observed data class : Class Data Record

The Observed Data class has only one record. This record is overwritten on command of the class that calls it, which means that only one observed data set is in memory at any given time.

type :: OBdata_Type

! The route with which the observation point is associated.

INTEGER :: Route

! The years in which the data starts and ends

INTEGER :: YStart, YEnd

```

! The observed flows values

REAL      :: QOBS(MNTPS)

! The observed flows codes.

CHARACTER(LEN=1) :: OBSC(MNTPS)

end type OBdata_Type

```

6.2.2 The Observed data class: Class Variables

One instance of the OBdata_Type called OD_Data This instance is used throughout.

6.2.3 The Observed data class: Methods

6.2.3.1 Public methods:

Public methods can be called from any other module or function that contains the statement

```
use obsdata_module
```

OD_GetCode

Retrieve the patching code from the OBSC array in the class data record OD_Data for the specified time period.

OD_GetValue

Retrieve the value from the QOBS array in the class data record OD_Data for the specified time period.

OD_GetYearEnd

Return the year in which the last observed value was recorded. It is the year in which the last value was recorded in the data file of the observation point that is currently loaded.

OD_GetYearStart

Return the year in which the first observed value was recorded. It is the year in which the first value was recorded in the data file of the observation point that is currently loaded.

OD_Init

Initialise the OD_Data record

OD_Invalidate

Invalidate the observation point data. It is used to force the function OD_Loaded to return FALSE so that the file with the data will be read again. So far, this function is only used by OB_Edit_GS when the name of the data file is changed.

OD_Loaded

Check whether the observation point associated with route with the USER DEFINED route number supplied is currently loaded into the Observed data class record OD_Data.

OD_Read

Read the designated observed flows data file into the OD_Data record and save the route, the beginning and end year of the data. Note: this function should ONLY be called by OB_READ and OB_Transfer. Handle both WRSM2000 .ANS or SpatSim .TXT files.

6.2.3.2 *Private methods:*

Private methods can only be called from within the module.

OD_GetVersion

Determine the version number for the input file specified. Return:

0 if this is an unknown file format

1 if this is a WRSM/Pitman input file (.ANS format)

2 if this is a SPATSIM input file

OD_SetValue

Set the value and the patch code that were read from the file in the appropriate places in the appropriate arrays in the class data record. Used in OD_Read.

7 THE HYDROLOGICAL MODULE CLASSES WITHIN WRSM/PITMAN

7.1 The Raingauge Class

The raingauge class is one of the basic classes in the program. This is because all the other network modules (i.e. the Runoff Area Module, the Reservoir Module, etc.) will use an instance of this class. The raingauge class contains the rainfall data series, the name of the file from which that data was read, the number of times this raingauge is being used and the MAP that was last associated with this raingauge.

It was decided that since a particular raingauge can be used in more than one network module, steps should be taken to stop the program reading the data for the same raingauge more than once. This strategy would help to save on memory space, and hence allow the loading and calibration of bigger networks.

In a new network, when a Network module (for example a Runoff Module) needs to load Raingauge data, it, in effect creates a new instance of the class Raingauge.

When the data for a raingauge has been read in once, this same gauge is not read in again. Instead, we save the number of times that the raingauge is used in different modules. If 4 modules in a network use the raingauge with the EN number 1, then the variable NTUsed in the Object Data Record of Raingauge 1 will be 4. If the user deletes one of the modules (for example one of the irrigation blocks) that uses the raingauge, the variable NTUsed is decremented by one. When the variable NTUsed is 0, the entire object data record is deleted.

The M.A.P. (Mean Annual Precipitation) of the Raingauge is stored within the Raingauge class. The reason for this is that the M.A.P. is used as a check.

Rainfall time series data is stored as percentages of M.A.P. and not in millimeters. When rainfall data is needed, the program converts these percentages to millimeters.

In some cases, the same Rainfall sequence data is used in more than one Module in a network. An Irrigation Block that lies within a Runoff Area must be modelled as lying outside of it. In this case it is then both convenient and correct to use the same Rainfall time series for both Modules.

It is possible to enter a different M.A.P. for the same Rainfall time sequence in different modules. In some cases this would be feasible – e.g. where a Runoff area is split into a more mountainous and a more level area or an Irrigation Block lies in a marginally drier portion of a Runoff area. The rainfall time sequence would be the same, but the actual precipitation would be different in the two modules. When the program reads the data and detects such an M.A.P. discrepancy, it cannot "know" whether this was done deliberately or whether the user made a mistake when he entered the data. For this reason, there is a check in the Raingauge class to determine whether the M.A.P. that is used for a given raingauge remains constant.

If the raingauge is used with M.A.P. of (say) 1000mm in one module and the same raingauge is used with a M.A.P. of (say) 900mm in another module, the program will ask whether this was intended. The user then has the choice to make the new M.A.P. that was detected (and which caused the warning message) the default M.A.P. to use for the next set of checks. Adjusting the M.A.P. in the raingauge only affects the check and not the actual calculations since the M.A.P. that is stored in the module is used for the calculations, and not the M.A.P. that is stored in the Raingauge class.

7.1.1 The Raingauge class: class data record.

```
type :: Raingauge_Type
```

```
! External Raingauge number (The user/program defined raingauge number)
```

```
INTEGER :: NRGEN
```

```
! Whether the raingauge data has been changed since it was read
```

```
! from the file
```

```
LOGICAL :: CHANGED
```

```
! Whether the raingauge is valid
```

```
LOGICAL :: VALID
```

```
! The number of times that the raingauge is used.
```

```
INTEGER :: NTUSED
```

```
! The raingauge name
```

```
CHARACTER(len=20) :: RGNAM
```

```
! Indicator to show that the raingauge data file has been read
```

```
INTEGER :: IGR
```

```
! File Name with the rainfall as percentage of MAP
```

```
CHARACTER(len=MNFILC) :: RGFile
```

```
! Mean Annual Precipitation
```

```
REAL :: CHMAP
```

```
! The raingauge data (as percentage of MAP)
```

```
REAL :: QRG(MNTPS)
```

```
end type Raingauge_Type
```

Supplementary notes:

The variable CHANGED is strictly speaking superfluous. There is no editing facility for raingauge data in the present program and so this variable (which indicates whether the data was changed) will never be set to true. In later versions of the program this could, however, change.

The variable NTUSED (i.e. Number of Times USED) also comes into play when a module is deleted. When a module that uses a particular raingauge is deleted, the value of NTUSED is decreased by 1. If the variable NTUSED is not 0, it means that other module(s) in the network still use this gauge. When they, too, are deleted and NTUSED becomes 0, the entire record for this raingauge is deleted.

7.1.2 The Raingauge class: class variables

MNRGS - The number of Raingauge Class Records that have been created.

NRG - The number of Raingauge Class Records that have been used so far.

In order to save time during execution, three new class data records are created at a time when a new record is needed. When all of them are used up and more records are needed, a further set of 3 records is created.

7.1.3 The Raingauge class: methods

7.1.3.1 Public methods

Public methods can be called from any other module or function that contains the statement

```
use raingauge_module
```

RG_Delete

Delete the specified raingauge class data record from memory.

RG_DeleteAll

Delete all raingauges from memory. Uses the method RG_Delete

RG_Edit

Edit raingauge data. Placeholder (dummy) method where the data (percentage of MAP) for the raingauge can be edited. Note that the data cannot be changed at this stage.

RG_GetFileN

Retrieve the name of the file from which the specified raingauge data was read.

RG_GetName

Retrieve the name of the Raingauge.

RG_GetMAP

Retrieve the MAP as it is currently stored for this particular rainfall station. Note: this M.A.P. is used for comparison purposes only to detect possible data errors and is not used in the actual calculations for the individual modules. The M.A.P. that is stored in the individual modules (e.g. Runoff, Irrigation Block, Channel Reach, etc.) is used for the flows calculations, since it is feasible that the same time sequence could be used for different modules with different M.A.P.'s.

RG_GetValue

Retrieve the raingauge data (as percentage of MAP) for the specified time period from the QRG array of the specified raingauge.

RG_IsSpatSimFile

Check whether the specified file is a SpatSim .txt rainfall file

RG_Read

Read the data into the appropriate Class Data Record. The method uses the RG_New method to create and obtain the index number of a class data record, before it transfers the data into this record. The method returns the unique number that identifies this raingauge (the EN number)

RG_SetMAP

Attempt to set the M.A.P. for a specific raingauge. If the MAP is different from the M.A.P. that was specified before (when another module was read, for example), the method issues a warning – see RG_GetMap.

RG_SetValue

Set the raingauge data (as percentage of MAP) for the specified time period into the QRG array of the specified raingauge. Note that this method is never called at this stage, but was included for the sake of completeness.

RG_ShowAll

Display the numbers of all rainfall stations, their names and the number of times that this raingauge is used by different modules.

7.1.3.2 Private methods

Private methods can only be called from within the module.

RG_GetIndex

Retrieve the index number in the array of Class Data records for the raingauge with the specified raingauge (EN) number

RG_NextNewEN

Determine the next (lowest) number for a new external number (EN).

RG_Init

Initialise newly created Class Data Record(s). Zeroise all arrays and sets all indicators to 'harmless'

RG_FFormat

Determine the format of the supplied Rainfile. If the format is incorrect or unknown, issue an error message.

RG_New

Create a new Class Data Record. Used by RG_Read when a new raingauge Class Data Record is required. If unused Class Data Records are available, use one of them. If no unused Class Data Records are available, attempt to create 3 additional ones and call RG_Init to initialise them. Return the index number of the Class Data Record to use, or 0 when all fails.

7.2 The Observation Point class

Observation Points are points where historical (i.e. Observed) flows can be compared to the flows that are simulated by the program. An Observation Point is situated on a Route. An Observation Point can therefore be visualised as a gauging weir.

Even though every Route can have an Observation Point associated with it, in practice this is not usually the case – only a small proportion of all routes will have Observation Points on them. It would therefore be a waste of memory to associate an Observation Point with every route as it is created. For this reason we reserve memory space for an Observation Point only when it is needed.

All Observation Points share a single Observed data structure. This structure was placed in its own class since there are certain functions that should work on this data. The speed with which computers work has increased significantly over the last few years, and so it was decided to load the data 'on the fly' when needed, rather than to keep all data in memory all of the time. The user will not notice the small delay while the data is loaded once again when it is needed. Again this strategy saves on memory space in the computer. As soon as one Observation Point is defined, there will be enough memory to load the data when it is needed.

The route with which an observation point is associated only knows the NOBEN (the unique ObjectEN number) of the Observation Point that is associated with it. Once the Observation Point is associated with a particular the Route, the Route is told only this number. Since it is none of the Route's business, it does not even know the file in which the historical data is kept – that is the job of the Observation Point class.

When a Route is deleted, any Observation Point that may be associated with it will be deleted as well.

7.2.1 The Observation Point class: Class Data Record

```
type :: ObsPoint_Type
```

```
! The number of this observation point.
```

```
INTEGER :: NOBEN
```

```
! Whether the observation point data has been changed since it the
```

```
! network was read from the file
```

```
LOGICAL :: CHANGED
```

```
! Whether the observation point is valid
```

```
LOGICAL :: VALID
```

```
! Indicator to show that the data file was checked and correct
```

```
INTEGER :: IOB
```

```
! The observation point name
```

```
CHARACTER(len=20) :: OBNAM
```

! File Name with the observed data

```
CHARACTER(len=MNFILC) :: OBFile
```

! The external route number with which this Observation point is

! associated.

```
INTEGER :: NOBRT
```

! The start year of data in the file, i.e. the first year for which there

! is data

```
INTEGER :: YStart
```

! The end year of data in the file, i.e. the last year for which there is

! data

```
INTEGER :: YEnd
```

```
end type ObsPoint_Type
```

Additional notes:

The variable CHANGED is really superfluous, but was added just in case we ever want to be able to edit the data.

7.2.2 The Observation Point class: Class Variables

MNOBS - The number of Observation Point Class Records that have been created.

NOB - The number of Observation Point Class Records that have been used so far.

7.2.3 The Observation Point class: Methods

7.2.3.1 Public Methods

Public methods can be called from any other module or function that contains the statement

```
use obspoint_module
```

OB_Delete

Deletes the specified Observation Point Class Data Record.

OB_DeleteAll

Deletes all the Observation Point structures that are currently created.

OB_Edit

Edit the data for the Observation Point with the specified external number (mm).

The method returns 1 if the editing was successful, or -mm to show that the user entered an invalid or 0 file name for the observation point. In the case where a negative mm is returned the observation point must be deleted by the calling routine, since it is not valid any more.

OB_Exists

Check whether an observation point with the specified external number exists. Return true or false.

OB_GetFileN

Return the name of the observations file associated with the specified Observation Point.

OB_GetName

Return the name of the specified Observation Point in a string of 20 characters.

OB_GetRoute

Report the user defined number of the route with which the specified observation point is associated.

OB_GetYearEnd

Return the last year for which there is data in the observed flows file of this Observation Point.

OB_GetYearStart

Return the first year for which there is data in the observed flows file of this Observation Point.

OB_HasChanged

Check whether any of the gauging stations for which there are class data records have been changed. Don't ask for the changes individually since if one of them has changed, the entire network file must be rewritten anyway.

OB_HowMany

Report how many Observation Points there are in the Network at present.

OB_Read

Read the file of observed data for checking purposes only. The observed data is not stored but the file name and the start and end years saved. The function OB_Transfer places the data from the file into the Observed data class record that can be accessed by other functions. Observed flows files are usually short – it will cost very little time but save a lot on memory when data is not stored when not in use.

OB_Saved

Tell the specified Observation Point that it has been saved in the Network file.

OB_SetName

Set the name of the specified Observation Point

OB_ShowAll

This method displays all the Observation Points that are currently in the network.

OB_Transfer

Instruct the specified observation point to transfers data from its observed flows file to the Observed data Class.

7.2.3.2 *Private Methods*

Private methods can only be called from within the module.

OB_GetIndex

This method returns the index number of the array of Observation Point class data records where the Observation Point with the external number mm can be found.

OB_NextNewEN

Determine the next (lowest) number for a new external number (EN).

OB_Init

Initialise a newly created observation point class data record.

OB_New

Assign a new Observation Point Class Data Record and returns the internal number of the record. If there is no class data record available to assign i.e. NOB equals MNOBS then the function attempts to assign a new class data record. Return the index number of the new record, or 0 if process fails.

OB_Edit_GS

This method sets or saves the values for the Edit dialog for the specified observation point.

7.3 The Route Class

A route is a dimensionless entity that serves to connect two Hydrological modules.

A route is therefore defined as running between an upstream module and a downstream module. When a route is defined, the upstream and downstream module numbers become part of the definition of the route.

The number of a given route will be used in both the upstream and downstream modules as well. In the upstream module to a route, the route will be defined as an outflow route and in the downstream module the route will be defined as an inflow route.

When the upstream module of a route is deleted, the upstream node number of the route is set to undefined. If the downstream module of the route is deleted, the downstream node number is set to undefined. If both the upstream and downstream modules of a route are deleted, both the upstream and downstream node numbers of the route are undefined, and therefore the route itself is deleted.

In addition to the upstream and downstream node numbers, every route has an internal and a user defined

number. If required, the route may also be associated with an observation point (see above). An observation point may be deleted without further effect to the route.

Routes should not be confused with Channel Modules. There can be no abstractions or inflows from other routes into a given route, and no losses be defined in a route. A route is a lossless, dimensionless entity where water flows unimpeded and without any time delay from one module into the next.

A Channel Module, on the other hand, may be used to combine the outflows from two or more routes into a single outflow route, and abstractions may occur here. In addition, it is possible to specify losses as a result of wetlands and seepage into the river bed in a Channel Module. A route does not cater for any of these abstractions or losses.

7.3.1 The Route Class: class data record

The Route data record. This structure contains all data to do with a Route.

type :: Route_Type

! External Route number (The user defined route number)

INTEGER :: NRTEN

! Whether the data was changed since it was read from the file

LOGICAL :: CHANGED

! Whether the data for the route is valid

LOGICAL :: VALID

! File Name for defined flows – MNFILC is a global constant.

CHARACTER(len=MNFILC) :: RTFile

! External (or user defined) upstream module number

INTEGER :: NUP

! Upstream module code – RU, CR, RV, RR

CHARACTER (len=2) :: NUPCODE

! Has the file name been changed but not yet saved by the

! upstream module? See RT_FNChanged

LOGICAL :: NUPFN

! External (or user defined) downstream node number

INTEGER :: NDN

! Downstream module code

CHARACTER (len=2) :: NDNCODE

! Has the file name been changed but not yet saved by the
! downstream module? See RT_FNChanged

LOGICAL :: NDNFN

! Indicator to show that route flow was read from a file:

! 1 if flow defined, 0 if the route is to be calculated

INTEGER :: IDR

! Indicator to show that this is a supply route to an irrigation
! block, and that the irrigation block will set the demands

INTEGER :: IIR

! Indicator to show that this route is a special route (or not),

! e.g. a local in- or outflow to a Wetland and which can therefore not be

! used for certain other purposes, such as being the primary outflow route

! to a channel reach module.

INTEGER :: ISR

! Indicator to show that flow in route has been calculated:

! 1 if a route has been done

INTEGER :: ICR

! The number of the observation point that is associated with this

! route (IOBS = 0 if there is no observation point)

INTEGER :: IOBS

! 1 if the monthly flows are set by a reservoir module, else 0

INTEGER :: KRVDEM

! Monthly demands on reservoir by this route (Set from a Reservoir module

! file or by editing an outflow route from a reservoir.)

REAL :: RVDEM(12)

! The flows in the route after simulation. MNTPS is the number of time periods (months) that
! can be simulated

REAL :: QRT(MNTPS)

! The demands on this route – this can be either the specified
! flows, the irrigation demands or any other demands that the route
! is supposed to carry. When the model is initialised just before
! it is run, the demands are copied to QRT. Once the model has run,
! the values in QRT will be those that were supplied by the simulation.

REAL :: QDEM(MNTPS)

end type Route_Type

7.3.2 The Route Class: class Variables

MNRTS - The number of route data records that have been created so far.

NRT - The number of route data records that are in use, so far

7.3.3 The Route Class: Methods

7.3.3.1 *Public methods*

Public methods can be called from any other module or function that contains the statement

use route_module

RT_AddObsPoint

Add the number of an observation point (Gauging Station) to the route

RT_CheckAll

Check all routes for validity

RT_DefOrCalc

Determine whether a particular route has been either defined or calculated.

RT_DeleteAll

Delete all routes – calls RT_Delete for every route.

RT_Edit

Call up the edit dialog

RT_Exists

Check whether a specified route exists.

RT_FNChanged

Determine whether the file name which was used to read the defined data to this route was changed, i.e. whether the file containing the defined data was reread.

RT_FNSaved

Check whether the file name for defined route flows was saved

RT_GetFileN

Return the file name

RT_GetMDem

Get the monthly defined route flow for a particular month

RT_GetNUPString

Get the Upstream module number and code as a string.

RT_GetNDNString

Get the Downstream module number and code as a string

RT_SetNatFlowFlag

Sets the naturalised flows flag for the route.

RT_GetNDNCode

Get only the Code (RU, CR, RR or RV) of the Downstream module

RT_GetObsPoint

Return the number of the Observation Point (Gauging Station) that is associated with a particular route.

RT_GetSortedEN

Get Sorted External Number. This function returns the USER DEFINED or EXTERNAL number of the route in the sorted external number array.

RT_GetSortedRS(index)

Get sorted Route String This function returns a 48 character long string in which there is

1. the external route number
2. the upstream module
3. the downstream module

The message index is the index number in the sorted array.

RT_GetSpecial

Get the Special code for this route.

RT_GetValue

Get the simulated Value of the specified route for the specified time period

RT_HasDefFlows

Determine whether a route contains defined flows.

RT_HowMany

Determine the number of routes that are currently in the Network

RT_NextNewEN

Determine the next feasible External Number that can be used for a Route.

RT_Read

Read the defined flows from a file. Create the route if it does not exist. If no defined flows are specified, create the route and zeroise the route flows.

RT_Reset

Reset the demands that were placed on the route, prior to a run for a simulation

RT_SaveFlows

Save the specified flows to a file

RT_SetDValue

Set the specified value for the specified route and time period in the QDEM array.

RT_SetMDem

Set the monthly demands in the QDEM array.

RT_SetNDN

Set the number of the downstream Module in the specified route.

RT_SetNUP

Set the number of the upstream Module in the specified route.

RT_SetValue

Set the simulated value for the time period in the specified route (in the QSUP array)

RT_SetSpecial

Set the 'is a special route' code for the specified route.

0 – not special

1 – local inflow or outflow to a wetland

2 – diversion route from a channel reach

RT_ShowAll

Show all the routes on the screen.

RT_WasCalculated

Determine whether the specified route was calculated or, if the route was defined, whether the defined values were transferred to the QSUP array.

RT_WriteSummary

Write the data for the specified route to the Summary file.

RT_DelObsPoint

Remove the reference to a Gauging Station from the specified routes.

RT_Zeroise

Sets only the simulated values for the route specified as mm back to 0.00.

7.3.3.2 *Private methods*

Private methods can only be called from functions within the class.

RT_CanDelete

Check whether a route can be deleted. Routes may only be deleted when they are orphans, i.e. they have neither upstream nor downstream modules and if it has been saved in the event that the defined data has been changed.

RT_Check(mm,icode)

Check the route mm for validity. Returns 1 if the route is OK or 0 if it isn't.

mm is the user defined route number. icode indicates whether a message should be issued if the route is OK. If icode is 'Y' then a message is shown on the screen otherwise the function closes silently.

RT_Delete

This is the destructor function of the Route data structure. The function deletes everything to do with the Route known by the external number mm. It returns 1 if the deletion was successful, or 0 when it was not.

RT_Edit_GS(nr,icode)

Set or saves the values for the Edit dialog nr is the INTERNAL number of the Route. The message icode has 3 states: 0 – clear the fields, 1 – set the values from what is currently in memory or 2 – save the values from the dialog to the memory.

The function returns 1 if things went OK or 0 if there was an error, especially when saving.

RT_EHeader(irtn)

This routine creates the top part of an error message. Error messages always start as:

Route: xxxxxx (the number of the route)

irtn is the internal route number (the record number) of the route.

RT_GetIndex(mm)

Gets the index number of the array of route data records for the route with the external number mm. The function returns the number of the index, or 0 if it could not be found.

RT_GetVersion

Get the version number for the input file to read. The function returns 0 if this is an unknown file format, 1 if this is a WRSM/Pitman input file (.ANS format), 2 if this is a SPATSIM input file

RT_GetSEN(nr)

Return the position of the record nr in the array of data records sorted by external number. nr is the index to the record in the array of routes whose position within the sorted array must be determined.

RT_GetSortedRI(index)

Get Sorted Route Record Index. Retrieve the sorted route number indicated by the index number. Index is the element number to be retrieved. In effect, what the function does is to sort the routes according to their external route numbers. Once it has done that, it returns the number of the record that would correspond to the index number of that sorted array that is required. This is a little tricky, therefore an example: Suppose we have 4 route records with the following external numbers:

record 1-12
record 2- 8
record 3- 6
record 4-1

if we sorted that on external numbers, the records would be ordered:

4, 3, 2, 1 which corresponds to external numbers 1, 6, 8, 12

If we call RT_GetSortedRI with the argument 2, the function would return a value 3, because record 3 contains the second smallest value for the external numbers.

RT_HasChanged

Check whether the data in the route was changed. Return true if the data in the module was changed after the file was read otherwise return false

RT_Init

Initialise all the variables in the Route record signified by the index number 'index' which is the index number in the class records array. The function returns 1 if it completed successfully or 0 if it failed.

RT_IsValid

Check whether the data in the route is valid. Return true if the data is valid, otherwise return false.

RT_New

Assign a new route structure and return the internal number of the structure. If there is no structure available to assign i.e. NRT equals MNRTS then the function attempts to assign a new data structure.

7.4 The Runoff Module Class

The Runoff Module class is the heart of WRSM/Pitman. The module calculates the runoff that is generated by a catchment that is modelled by means of WRSM/Pitman.

The Runoff Module class contains a number of models that can be switched on or off as required.

It contains:

- a surface runoff model;
- soil moisture/groundwater models;
- afforestation models;
- an alien vegetation model and
- a paved area model.

The class uses the Raingauge class to keep the rainfall data that is used in the calculations. It also uses the Route class to accept a single inflow route from upstream and up to 10 outflows routes to store the simulated flows.

7.4.1 The Runoff Module Class: class data record

This structure contains all data to do with a Runoff module.

type :: Runoff_Type

! Runoff submodel number (external)

INTEGER :: NRUEN

! Runoff module data file name


```

CHARACTER(LEN=MNFILC) :: RUFILC
! Indicator to show that the data for this module
! has been changed since it was read from the file.
LOGICAL :: CHANGED
! Indicator to show that the data for this module is valid
! If all data has been checked, VALID will be set to TRUE
LOGICAL :: VALID
! Runoff submodel name
CHARACTER(LEN=20) :: RUNAM
! Number of runoff exit routes
INTEGER :: NRURR
! Runoff submodel exit route numbers (external numbers)
INTEGER :: NRURRN(MNRUOUT)
! Percentage of runoff via a particular exit route (above)
REAL :: RUPERC(MNRUOUT)
! Coefficient in soil moisture / subsurface flow equation.
REAL :: A
! Catchment area in km^2
REAL :: AREA
! Evaporation coefficient
REAL :: B(12)
! Evaporation coefficient
REAL :: C(12)
! Fraction of month represented by time step
REAL :: DT
! Current groundwater flow in mm/month
REAL :: FG1

```

```

! Current interflow in mm/month

REAL :: FI1

! Forest factor

REAL :: FF

! Subsurface flow at full soil moisture capacity

REAL :: FT

! Groundwater fraction of subsurface flow

REAL :: GFRACT

! Lag of groundwater flow

REAL :: GL

! Maximum groundwater flow in mm/month

REAL :: GW

! Number of time steps per month (default=4)

INTEGER :: NOFT

! Number of years for forested areas

INTEGER :: NYRF

! Afforestation years with the area (max MNRUFOR)

type(YearValue_Type) :: Forest(MNRUFOR)

! Number of years for afforestation (for editing)

INTEGER :: TNYRF

! Afforestation years with the area (max MNRUFOR) (for editing)

type(YearValue_Type) :: TForest(MNRUFOR)

! The number of the afforestation algorithm to use:

! 0 for Van der Zel curves

! 1 for CSIR

! 2 for Smoothed Gush

! 3 for LUT Gush

! 4 for User Defined reduction

```

INTEGER :: FORESTALG

! The number of the vegetation algorithm to use

! 0 for none,

! 1 for CSIR

! 2 for Riparian vegetation

INTEGER :: VEGETATIONALG

! The number of the Groundwater algorithm to use

! 0 for the classic Pitman model

! 1 for the Hughes model

! 2 for the Sami model

INTEGER :: GWMODEL

! Number of paved area data pairs

INTEGER :: NYRP

! Paved area years with the area as proportion of the catchment (max MNRUPAV)

type(YearValue_Type) :: Paved(MNRUPAV)

! Number of paved area data pairs (for editing)

INTEGER :: TNYRP

! Paved area years with the area as proportion of the catchment (max MNRUPAV)

! For editing

type(YearValue_Type) :: TPaved(MNRUPAV)

! Number of years for alien veg. areas

INTEGER :: NYRA

! Alien veg. years with the area (max MNRUAL)

type(YearValue_Type) :: Alien(MNRUAL)

! Number of years for alien veg. (for editing)

INTEGER :: TNYRA

! Alien veg. years with the area (max MNRUAL) (for editing)

type(YearValue_Type) :: TAlien(MNRUAL)

! Indicator to show whether to incorporate Alien Vegetation in the
! simulations if ALIENVEG is not 0, it will be used (if there is data, of
! course)

INTEGER :: USEALIENVEG

! Raingauge number.

INTEGER :: NPU

! Rotation period of 3 main afforestation types

INTEGER :: IROT(3)

! Mean monthly potential evapotranspirations in mm

REAL :: PE(12)

! Subcatchment MAP in mm

REAL :: PMEAN

! Current soil moisture storage status in mm

REAL :: Q1

! Current soil moisture storage status in mm

REAL :: S1

! Total catchment runoff in mm

REAL :: TFLOW

**Pitman Parameters for the currently selected Groundwater Model – these are set by the
RU_LoadParameters method and change as a different groundwater model is selected.**

! Power in soil moisture / subsurface flow equation

REAL :: POW

! Soil moisture state when subsurface flow = 0

REAL :: SL

! Soil moisture capacity in mm

REAL :: ST

! Subsurface flow at full soil moisture capacity

REAL :: FT

! Maximum catchment absorption rate in mm/month

REAL :: ZMAX

! Maximum groundwater flow in mm/month

REAL :: GW

! Minimum catchment absorption rate in mm/month

REAL :: ZMIN

! Interception storage in mm

REAL :: PI

! Lag of flow (excluding groundwater)

REAL :: TL

! Lag of groundwater flow

REAL :: GL

! Coefficient in evaporation / soil moisture equation

REAL :: R

Arrays of Pitman Parameters, one for each groundwater model

! The array of POWs, one for each groundwater model.

REAL :: A_POW(NGWMODELS)

! The array of SL's, one for each groundwater model.

REAL :: A_SL(NGWMODELS)

! The array of ST's, one for each groundwater model.

REAL :: A_ST(NGWMODELS)

! The array of FT's, one for each groundwater model.

REAL :: A_FT(NGWMODELS)

! The array of GW's, one for each groundwater model.

REAL :: A_GW(NGWMODELS)

! The array of ZMIN's, one for each groundwater model.

REAL :: A_ZMIN(NGWMODELS)

```

! The array of ZMAX's, one for each groundwater model.
REAL :: A_ZMAX(NGWMODELS)

! The array of PI's, one for each groundwater model.
REAL :: A_PI(NGWMODELS)

! The array of FF's, one for each groundwater model.
REAL :: A_FF(NGWMODELS)

! The array of TL's, one for each groundwater model.
REAL :: A_TL(NGWMODELS)

! The array of GL's, one for each groundwater model.
REAL :: A_GL(NGWMODELS)

! The array of R's, one for each groundwater model.
REAL :: A_R(NGWMODELS)

! The array of GPOWs, one for each groundwater model.
REAL :: A_GPOW(NGWMODELS)

! Whether or not to produce naturalised outflows
LOGICAL :: RunNaturalised

! Monthly pan evaporation
REAL :: SPAN(12)

! Monthly pan factor
REAL :: SPF(12)

! Percentage of afforestation under 3 main types
REAL :: RPERC(3)

! Percentage of forest in optimal area
REAL :: RPOT

! Percentage of alien vegetation under 3 main types
REAL :: APERC(3)

! Age of alien vegetation types
REAL :: AGE(3)

```

! Percentage of alien veg. in optimal area

REAL :: APOT

Specific for the Hughes Groundwater Model

! The Hughes Groundwater model runs twice – once to warm up and once for real

INTEGER :: RunNumber

! Whether this runoff module has already been 'run' – will be .FALSE. until

! the module ran without a hitch.

LOGICAL :: WasSimulated

! The Runoff Module that lies upstream of this one

INTEGER :: UpStrRunoffModule

! The cumulative area of the runoff modules in the network up to (and

! including) this module

REAL :: CUMA

! The length of a drainage slope element

REAL :: DSLength

! The number of Drainage reaches

INTEGER :: NDRAINS

! The drainage density

REAL :: DRAINAGEDENSITY

! The total width of a drainage slope

REAL :: DSTotalWidth

! The percentage of the total slope that is the upper zone of

! the drainage slope.

REAL :: DSPerc1

! The percentage of the total slope that is the Riparian zone of

! the drainage slope.

REAL :: DSPerc2

```

! Drainage slope WIDTH for the Upper portion of the catchment

REAL :: DSWidth1

! Drainage Slope WIDTH for the Riparian portion of the catchment

REAL :: DSWidth2

! The number of drainage slopes. Was p[43] in the original code.

INTEGER :: NDSlopes

! Rest Water level – the level of water below the channel in m

! Is used as a negative value but is stored as a positive. This was p[37] in the original code.

! Always use this as (0.00 – RestWL)

REAL :: RestWL

! The Rest Water Level Gradient Was p[45] in the original code.

REAL :: RestWLGrad

! This is 70% of the RestWLGrad – used for testing purposes.

REAL :: MAXSL

! Maximum recharge rate / month in mm. This variable has a different

! meaning from GW in the original model, hence the new name. This is p[18] in the original
code.

REAL :: HGGW

! HGA is slightly different from A in that SL is always 0.00. It is named

! differently, because a user may swap between groundwater models.

!  $HGA = FT / (ST - 0.00) ** POW$ 

! In the original code, this is called p[41]

REAL :: HGA

! For the Hughes Groundwater model, the Power for the Storage Recharge curve.

! This is p[32] in the original code.

REAL :: GPOW

! The initial GW slope value.

REAL :: GWSLInit

```


! For the Hughes Groundwater model, the Storage Recharge Curve.

! HGSRC = HGGW / (ST – SL) ** GPOW

! This is p[42] in the original code.

REAL :: HGSRC

! The maximum flow in the channel in this catchment in any given month.

REAL :: MAXQC

! Temporary version of MAXQC to keep after the model ran once.

REAL :: TMAXQC

! The maximum runoff that will be generated by this catchment in any given
! month.

REAL :: MAXQS

! The temporary version of MAXQS to keep after the model ran once

REAL :: TMAXQS

! The maximum possible loss that can occur in the river channel
! (in mm) over the entire length of the channel. (Parameter)

! Was p[31] in the original code

REAL :: TLGMax

! The maximum loss that occurs in a channel

REAL :: TLGMaxc

! The Channel Loss

REAL :: CLOSS

! Flow1 and Flow2 are really local summation variables used in the Hughes
! Groundwater model – kept so that the original author recognizes his
! algorithm.

REAL :: FLOW1

REAL :: FLOW2

! qq1 is really the same as Q1, but kept so that the original author

```

! can recognize his algorithm.

REAL :: gq1

! The Groundwater abstractions in the Upper portion (MI/Year)

! Was p[39] in the original code.

REAL :: GWAbstr1

! The Groundwater abstractions in the Riparian portion (MI/Year)

! Was p[40] in the original code.

REAL :: GWAbstr2

! The monthly Groundwater abstraction demands in the Upper portion

! as proportion.

! gwdem in the original code.

REAL :: GWDemand1(12)

! The monthly Groundwater abstraction demands in the Riparian portion

! as proportion.

! Was gwdem in the original code.

REAL :: GWDemand2(12)

! Groundwater slope for the Upper portion

REAL :: GWSL1

! Groundwater slope for the Riparian portion

REAL :: GWSL2

! Groundwater volume for the Upper portion

REAL :: GWV1

! Groundwater volume for the Riparian portion

REAL :: GWV2

! The inflow route from upstream (only 1 allowed)

INTEGER :: RTIN

! No recharge below this storage.

REAL :: HGSL

```

```

! Riparian Strip Factor

! The riparian strip factor is the proportion of the slope element in the
! GW geometry system from which ground water can evaporate. This
! is to allow GW that is below the surface to go even lower (i.e.
! negative slope). It is a fraction and should probably
! not be much higher than 0.05 (in extreme cases). I have used
! values of 0.01 to 0.001. It will always be a difficult parameter to
! establish, but is necessary. It could be interpreted as the 'riparian
! evaporation zone'.

! was p[38] in the original code.

REAL :: RipStripF

! Transmissivity in m2/day

! Was p[34] in the original code

REAL :: Transmiss

! Storativity

! Was p[35]in the original code

REAL :: Storativity

! Storing recharge for month (mm)

REAL :: RCH

! RC1

REAL :: RC1

REAL :: SQ1

! The Groundwater state that is delivered to the next downstream
! runoff module.

REAL :: GWToNextRU(MNTPS)

```

Specific for the Sami Groundwater Model

```

! The monthly groundwater abstractions

```

```

LOGICAL :: AbstractionsOn

! The Sami Model to use (if it is used, that is)

INTEGER :: SamiModel

REAL :: AD35_GWAbstraction(12)

! The aquifer thickness

REAL :: B8_AquiferThick

! The storativity

REAL :: B9_Storativity

! The aquifer capacity

REAL :: B10_AquiferCap

! The Aquifer storage

REAL :: B11_AquiferStor

! The static water level

REAL :: B13_StatWaterLevel

! No recharge below this storage.

REAL :: B17_HGSL

! The maximum discharge rate

REAL :: B21_MaxDischRate

! The Power in the Groundwater/surface water equation

REAL :: B22_GWSW_ICP

! B23 Maximum Hydrological gradient

REAL :: B23_MaxHydGrad

! B28 Groundwater evaporation area

REAL :: B28_GWEvapArea

! B28 Groundwater evaporation area / the total catchment area.

REAL :: B28_GWEvapAreaFactor

! The decay for Sami version 1

REAL :: D7_Decay

```

```

! The Unsaturated storage for Sami version 2

REAL :: D7_Unsat

! The growth percentage for Sami version 1

REAL :: D8_GrowthPerc

! The initial storage of the unsaturated zone Sami version 2

REAL :: D8_STinit

! The number of months over which the recharge must be averaged.

INTEGER :: D9_MMA

! The initial base flow for Sami version 1

REAL :: D13_InitBaseFlow

! The percolation power PPOW for Sami version 2

REAL :: D13_PPOW

! Maximum recharge rate / month in mm. This variable has a different
! meaning from GW in the original model, hence the new name.
! This is D15 in the original code.

REAL :: D15_HGGW

! GPOW

REAL :: D16_GPOW

! The transmissivity

REAL :: D21_Transmissivity

! The distance of the boreholes to the river

REAL :: D22_DistToRiver

! The maximum from groundwater percentage (always 100)

REAL :: D23_MaxFromGWperc

! The parameter K2

REAL :: D24_K2

! The parameter K3

```

```

REAL :: D25_K3

! The parameter F28 the interflow lag

REAL :: F28_Lag

! Number of abstraction data pairs

INTEGER :: NYGA

! Paved area years with the area as proportion of the catchment (max MNRUGWA)

type(YearValue_Type) :: GWAbst(MNRUGWA)

! Number of groundwater abstraction data pairs (for editing)

INTEGER :: TNYGA

! Groundwater abstraction years with the abstraction in Mm3 (max MNRUGWA)

! For editing

type(YearValue_Type) :: TGWAbst(MNRUGWA)

!

! The intermediate time series

!

! The outflow as a result of Runoff from this Module

! excluding Paved area runoff and upstream inflow.

REAL :: TS_RunoffOutflow(MNTPS)

! The outflow as a result of Runoff from this Module

! via the surface, i.e. paved area runoff and component f (the surface flows)

REAL :: TS_SurfaceOutflow(MNTPS)

! The total paved outflow Time Series

REAL :: TS_PavedOutflow(MNTPS)

! The riparian vegetation demand

REAL :: TS_RipVegDemand(MNTPS)

! The Sami Groundwater storage

REAL :: TS_SamiGWStorage(MNTPS)

```

! The groundwater outflow

REAL :: TS_GroundwaterOutflow(MNTPS)

! The groundwater storage

REAL :: TS_GroundwaterStorage(MNTPS)

! The outflow that is attributable to the inflow into the module

! from upstream. This will be 100% in the case of the Pitman model, but

! may be less if the user specified the Hughes model.

REAL :: TS_FromupOutflow(MNTPS)

! The pervious area total (ground and surface water) average runoff depth

REAL :: TS_PATotalROD(MNTPS)

! The pervious area surface water average runoff depth

REAL :: TS_PASurfROD(MNTPS)

! The recharge in mm

REAL :: TS_Recharge(MNTPS)

! The average groundwater storage – this is the storage of the catchment

! when the catchment is a normal or Child storage, but the average

! area-weighted storage if the catchment is a Parent catchment.

REAL :: TS_AvGWStorage(MNTPS)

!

!

Note on the Sami method – Sami developed his method using a spreadsheet. Three loops were required in the WRSM/Pitman code to produce the same logic as the spreadsheet because the spreadsheet makes use of averaged values for certain parameters. The first loop calculates the average recharge, the second the average percolation and then the third loop recalculates everything using the averages previously determined.

7.4.2 The SFR Parent/Child parameters

!

! ModSubType can have 4 states:

! 0 – Normal common and garden runoff module, not a Parent, not a Child

```

! 1 – the runoff module is currently a Parent module

! 2 – the runoff module is currently a Child module

! 3 – the runoff module is currently a covariate module (Not implemented yet)

INTEGER :: ModSubType

!

! When this catchment acts as a Parent:

! The number of Child catchments within this catchment

INTEGER :: NChilds

! The array with the external (user defined) module numbers of the

! catchments that are Childs to this catchment

INTEGER :: MyChilds(MNCHILDS)

!

! When this catchment is a Child catchment:

! The external module number of the Parent catchment of this catchment

INTEGER :: MyParent

! The type of SFR that is contained in this Child catchment

! 0 – None, 1 – CSIR forests, 2 – Smoothed GUSH, 3 – Look up Table Gush,

! 4 – User Defined reductions, 5 – CSIR Alien Vegetation

INTEGER :: NSFR

! If the module is a Child AND the module has User Defined reductions:

! User defined SFR reduction for MAR

REAL :: UD_TPRM

! User defined SFR reduction for Low Flows

REAL :: UD_TPRL

! The percentage of the Alien vegetation that is Riparian Alien Vegetation

! in an SFR

REAL :: RipVegAreaPerc

!

```


! Whether or not to include the riparian vegetation in the simulation

LOGICAL :: RunRiparianVeg

end type Runoff_Type

7.4.3 The Runoff Module Class: class Variables

MNRUS - The number of Object Data Records that have been created

NRU - The number of Object Data Records that have been used

CURRENTFILEVERSION - The file version number to use when writing files.

7.4.4 The Runoff Module Class: Methods

7.4.4.1 Public methods

Public methods can be called from any other module or function that contains the statement

use runoff_module

RU_AddInFlow

Add an inflow route to the specified module – called when adding a route that has a runoff module as downstream module.

RU_AddOutFlow

Add an outflow route to the specified module – called when adding a route with a runoff module as upstream module.

RU_Calculate

Run the simulation for the specified runoff module for the entire simulation time period. The function determines which groundwater model and (if specified,) which afforestation model to run and stores the final flows in the outflow routes.

RU_CanDelete

This function checks whether a Runoff module can be deleted. A Runoff module may only be deleted if it has been saved in the event that the data has been changed.

RU_Check

Check the data in the object data record for the specified runoff module

RU_CheckAll

Check the data in the object data record for all runoff modules in the system.

RU_CopyClimate

This method allows a CHILD module to copy all the climate data from its PARENT module.

RU_CopyParameters

This method copies all the CALIBRATION parameters from the runoff module with the EXTERNAL number “mmfrom” to the runoff module with the external module number “mmtto”.

The function returns 1 if it succeeded or 0 if it failed.

RU_Delete

Check whether no data will be lost by means of RU_Candelete, then (if that is OK) delete the specified runoff module from memory.

RU_DeleteAll

Delete all runoff modules from memory one by one using RU_Delete.

RU_DelInflow

Delete the inflow route from the runoff module – called when a route is deleted that has a runoff module as downstream module.

RU_DelOutflow

Delete the specified outflow route from the specified runoff module – called when a route is deleted that has a runoff module as upstream module.

RU_DoesNatFlows

This function reports whether the Runoff Module that is specified by means of the EXTERNAL module number mm has been set to produce Naturalised flows. The function returns TRUE if the module is to produce Naturalised flows or FALSE if it is to produce as-is flows.

RU_Edit

Set up the dialog and edit the values in the Object Data Record for the specified runoff module. This function calls the private function RU_Edit_GS to manage the reading and writing of the values in the dialog.

RU_Exists

This method checks whether a runoff module with the specified external number exists. It returns true if the runoff module exists, or false if it does not.

RU_GetFirstModuleEN

Retrieves the external number of the first Runoff module that is currently in the network. The function returns the first external number that it finds or 0 if there are no runoff modules in the network.

RU_GetGWModel

Retrieve the Groundwater Model that is used in the specified Runoff Module.

The function returns:

-1 for an invalid module number;

0 when the module uses the Pitman Groundwater Model and

1 when the module uses the Hughes Groundwater Model.

RU_GetMAP(mm)

This method allows other parts of the program to get the value of MAP for the specified module.

Where:

mm is the external module number of the module that must be interrogated;

the method returns the MAP of the module if successful or

0.0 (and user messages on the screen) when it failed.

RU_GetNumChilds

This function returns the number of Childs in the module with the external number mm.

The function can return 3 states:

-1 if the module mm is not a Parent module

0 if it is a Parent module with no Children or

n (where n is an integer) if the module is a Parent with n Children.

RU_GetRipVegDemand(mm, ntp)

This function returns the riparian vegetation demand for the specified time period in the specified runoff module.

Where:

mm – is the external number of the runoff module

ntp -- is the time period.

RU_GetSFReduction(mm, TPRM, TPRL)

This subroutine returns the target percentage flow reductions for MAR and for low flow for the Child module with the external number mm.

RU_GetChildModNumber(mm,n)

This function returns the EXTERNAL module number of the nth Child in the Parent module with the EXTERNAL module number mm.

Where:

- mm is the EXTERNAL module number of the Parent catchment and
- n is the index number of the Child whose external module number you want.

RU_GetChildOutRT(mm)

This function returns the outflow route to the SFR Child module. Where :

- mm – is the EXTERNAL module number of the Child;

Child modules may only have one outflow route;

the function returns the outflow route of the Child if it exists or

0 if the route was not defined or the module is not a Child.

RU_GetFF(mm)

This method allows other parts of the program to get the value of FF for the specified module.

Where:

- mm is the external module number of the module that must be interrogated;

The method returns the FF of the module if successful or

0.0 (and user messages on the screen) when it failed.

RU_GetHGSL(mm)

This method allows other parts of the program to get the value of HGSL for the specified module.

Where:

- mm is the external module number of the module that must be interrogated;

The method returns the HGSL of the module if successful or

(and user messages on the screen) when it failed.

RU_GetPI(mm)

This method allows other parts of the program to get the value of PI for the specified module.

Where:

- mm is the external module number of the module that must be interrogated;

The method returns the PI of the module if successful or

(and user messages on the screen) when it failed.

RU_GetST(mm, ntp)

This method allows other parts of the program to get the value of ST for the specified module.

Where:

- mm is the external module number of the module that must be interrogated

- ntp is an OPTIONAL time period number

The method returns the ST of the module if successful or

(and user messages on the screen) when it failed.

If the time period is specified, then the value that was stored in the time series TS_GroundwaterStorage is returned.

RU_GetSL(mm, n)

This function returns the EXTERNAL module number of the nth Child in the Parent module with the EXTERNAL module number mm.

Where:

- mm is the EXTERNAL module number of the Parent catchment
- n is the index number of the Child whose external module number you want.

RU_GetName

Retrieve the name of the specified Runoff module.

RU_GetPervRunoffDepth

Retrieves the runoff depth (in mm) for the pervious area of the specified runoff module for a given time period. This method is used by the WQT irrigation module routine where this value is known as qc.

RU_GetPervSurfaceRunoff(mm, ntp)

This function retrieves the surface runoff depth (in mm) for the pervious area of a runoff module for a given time period. The surface runoff excludes the groundwater runoff.

Where:

- mm is the EXTERNAL (i.e. user defined) number of the runoff module to interrogate
- ntp is the time period for which the runoff depth must be determined.

This method is used by the WQT mining module routine where this value is known as qveld.

RU_GetSortedRS

Return a 48 character long string which contains

1. the external runoff module number
2. the runoff module name

The argument index is the index number, i.e. the xth runoff module to be found in the sequence of sorted external numbers.

RU_HasChanged

Check whether the data in the specified runoff module was changed since the file was read or saved the last time. Return true if the data has changed, otherwise return false.

RU_HowMany

Determine the number of runoff modules that are currently in the network.

RU_HasCSIRForests

This function reports whether the Runoff Module specified by means of the EXTERNAL module number mm has afforestation data and uses the CSIR method to calculate the reductions as a result of this.

The function returns TRUE if there are CSIR forests or FALSE if there are not.

RU_HasCSIRVegetation

This function reports whether the Runoff Module specified by means of the EXTERNAL module number mm has alien vegetation data and uses the CSIR method to calculate the reductions as a result of this.

The function returns TRUE if there are CSIR alien vegetation area or FALSE if there is not.

RU_HasRiparianVeg

This function reports whether the Runoff Module that is specified by means of the EXTERNAL module number mm has riparian vegetation data that must be taken into account for stream flow reductions.

The function returns TRUE if there is Riparian vegetation in the Runoff module or FALSE if there is none.

RU_IsParent

This method can be used from outside the module to check whether the module with the EXTERNAL number mm is a Parent module.

The method returns TRUE if the module is a Parent or false if it isn't

RU_IsChild

This method can be used from outside the module to check whether the module with the EXTERNAL number mm is a Child module.

The method returns TRUE if the module is a Child or false if it isn't

RU_NextNewEN

Return the next (lowest) number for a new external number. Used to determine a suitable new module number.

RU_Param

Writes the calibration parameters for all the runoff modules to a table in the file that is opened as KOUT. The function is called by the function Result (in the stats_module) that writes the statistics.

RU_Print

Create a printout with all parameters for the specified runoff module.

RU_PrintAll

Print all the runoff modules in the network, one after the other using RU_Print.

RU_Read

Create a new Runoff Object Data record with the specified object number and read the data from the specified file. If no number is passed, create a new number by means of RU_NextNewEN. If no file name is specified, initialise all the data to harmless values, set the module as invalid and await editing.

RU_Reset

Reset the outflow routes to 0.00, recalculate (or reset) all the initial values that are needed in anticipation of a call to RU_Calculate for the specified runoff module.

RU_Save

Save the data and parameters for the specified runoff Module to the module data file.

RU_SaveModuleTS

This method displays the dialog that allows the user to select the Runoff module for which to save Module Time Series (internal storages, etc.) and the Time Series to save.

It then calls the function RU_WriteTS with the appropriate code.

RU_SetAllChilds

This function makes sure that all Children are informed that they are Children It is called ONLY after the entire network is read, OR when a runoff module has been imported.

RU_SetNaturalised(mm, onoff)

This method allows other parts of the program to set the way in which the simulation is run, i.e. with the class variable RunNaturalised either true or false.

Where:

mm is the external module number of the (Child) module that must be set onoff is either:

1 for naturalised simulation or

0 for a present day simulation.

RU_SetRiparianVeg(mm, onoff)

This method allows other parts of the program to set the way in which the simulation is run, i.e. with the class variable RunRiparianVeg either true or false.

Where:

mm is the external module number of the (Child) module that must be set onoff is either:

1 for include the riparian vegetation in the simulation or

0 for a simulation without the riparian vegetation.

RU_SetFF(MM, FFNew)

This method allows other parts of the program to set the parameter FF for the specified Child module.

Where:

mm is the external module number of the (Child) module that must be set

FFNew is the new value for the calibration parameter FF

RU_SetHGSL(MM, HGSLNew)

This method allows other parts of the program to set the parameter HGSL for the specified Child module.

Where:

- mm is the external module number of the (Child) module that must be set

HGSLNew is the new value for the calibration parameter HGSL

RU_SetPI(mm, PINew)

This method allows other parts of the program to set the parameter PI for the specified Child module.

Where:

- mm is the external module number of the (Child) module that must be set and

PINew is the new value for the calibration parameter PI

RU_SetSL(mm, SLNew)

This method allows other parts of the program to set the parameter SL for the specified Child module.

Where:

- mm is the external module number of the (Child) module that must be set and

SLNew is the new value for the calibration parameter SL

RU_SetST(mm, STNew)

This method allows other parts of the program to set the parameter ST for the specified Child module.

Where:

- mm is the external module number of the (Child) module that must be set and

STNew is the new value for the calibration parameter ST

RU_ShowAll

Display all the runoff modules that are currently in memory on the screen, together with an indicator to show whether the data was checked and is feasible.

7.4.4.2 *Private methods*

Private methods can only be called from functions within the class.

RU_ALIEN

Calculate the stream flow reduction due to alien vegetation.

Based on program alveg by Pitman and Bailey. It uses the results of the study by Forestek, CSIR reported in the WRC report "Invasive alien vegetation and dam yields – Illustrating the Impact of Assurance of Supply"

RU_Edit_GS

Clear, Set or Save the values for the Edit dialogs for the specified Runoff module.

Returns 1 on success or 0 if there was an error.

RU_EHeader(iru)

Create the top part of an error message in the Estring Class. Error messages always start: Runoff module xxxxxx (the name of the module) Module Number: yyy and then the rest of the message.

RU_FOREST

Calculate the stream flow reduction due to afforestation using the CSIR model.

This subroutine is based on program affdem3 by Allen and McKenzie. It uses the results of the study by Forestek, CSIR reported in the paper "Preliminary empirical models to predict reduction in total and low flows resulting from afforestation" (Water SA Vol 23 No. 2)

RU_FRLOSS

Read tables into a matrix, based on tree-type, rotation period in years, Afforestation reduction factors and a letter to indicate Pine, Eucalypt and Wattle for the CSIR afforestation algorithm.

Function RU_GetFinalGW(mm,ntp)

Retrieve the final groundwater state that is to be passed to the next runoff module downstream for the specified Runoff module for the specified time period.

Issue a warning message and return -1.00 when process fails.

RU_GetSFReduction

This subroutine returns the target percentage flow reductions for MAR and for low flow for the Child module with the external number mm

RU_SMGush

This routine gets the target % reductions from Gush's data for 843 quaternary catchments that has been generalised & smoothed by averaging for 3 soil depths and deriving linear relationships between MAP and % reduction (MAR & low flow) for same 3 tree types as CSIR.

RU_GetIndex

Get the index number of the array Runoff_PA where the specified Runoff module can be found.

The function returns the index number or 0 if a record for the specified user defined runoff module number could not be found.

RU_GetVersion

Obtain the version number of a file that contains data for a Runoff Module by doing a low-level read operation. Return the version number of the file, or 0 if no version number was found in the file. Since 0 is the first version number, this is OK. If the file has no data (or the data is wrong), issue a warning and return -1

RU_GW

Calculate the groundwater flows according to the method by Dr Pitman during the 4 timestep portion of RU_Calculate for the specified runoff module.

RU_GWCalcPhysical

Calculate the physical attributes that are needed for the Hughes groundwater model for the specified Runoff Module. Used by RU_Reset and RU_Edit_GS.

Set the class data structure variables NDSlopes, the number of drainage slopes, DSLength, the length of the drainage slope segments and DSTotalWidth, the total width of the drainage slope. Divided DSTotalwidth further into a Riparian- and an Upper Zone.

Return 1 if successful, or 0 if it failed.

RU_HGCalcs

Calculate the ground water discharge to the stream as gflow in mm for the specified Runoff module as part of the Hughes model groundwater calculations. Called from RU_Calculate.

Original: procedure tform1.gw_calcs The function returns 1 if it succeeded or 0 if it failed.

RU_HGMoist

Calculate the soil moisture state with the Hughes Groundwater model during the 4 timestep portion of RU_Calculate for the specified runoff module.

The returned values of sflow and gflow are the average of those estimated in this time interval (iteration) and the previous one – to allow smoothing during rapid changes in moisture status.

The function returns 1 if succeeded or 0 when it failed.

Function RU_Init

Initialise all the variables in the Runoff record for the specified index number within the array Runoff_PA Returns 1 if completed successfully or 0 if failed.

RU_New

Assign a new runoff structure and return the internal number of the structure. If there is no structure available to assign, i.e. NRU equals MNRUS then attempt to assign a new data structure. Return 0 if operation failed.

RU_PrintM(mm, KOUT1)

Write the data of the specified Runoff Module to the file opened at a specific Fortran unit number. The file must be opened already.

RU_QRTPERCENTILE(rnum)

The routine returns the value in the specified route number that is equaled or exceeded 25% of time for use in the CSIR afforestation calculations. Modified version of the original SORTR1.

Function RU_SetActives(ns)

Set tabs and fields either active or disabled depending on which one of the groundwater model types is in use for the module with the specified internal number. Returns 1 if the operation was successful or 0 if it failed. Works only on the dialog RU_Edit and its sub-dialogs.

RU_SPECIAL_AREA

Add all special areas within a given runoff module for a given year and compare that to the total catchment area of the Runoff module. Issue an error message when the special area is greater than the total catchment. Depending on a message in the call, return either the total special area, the area under afforestation only, the paved area only or the area under alien vegetation only.

RU_Write

Write the data of the specified Runoff Module to the input file that was used before. If no file was read, create a new file with a standard name in the input directory.

NOTE: Never call this function directly – additional checks must be made in RU_Save before this operation.

Function RU_IsValid

Return true if the data in the module is valid otherwise return false.

RU_WriteFlows(mm, tempf, iopt)

Write a time series to a file, automatically overwriting this file every time the simulation runs.

1. naturalised flows (save to .MRV file)
2. soil moisture (S) (save to .MRS file)

Note: Only option 2 (write soil moisture storages) works satisfactorily at this time, because it is not feasible to remove all manmade influences while running the simulation.

These public and private functions and links showing which functions call on other functions are shown in Figure 1 – Enhanced WRSM/Pitman Runoff Module Class.

7.5 The Irrigation Module Class

The Irrigation Module class is used to simulate the behaviour of an irrigation block. An irrigation block can be a run-of-river scheme or a reservoir fed scheme.

An irrigation block is a demand centre. As a first step, crop demands are calculated and these demands will be demanded from Irrigation Abstraction Route that is to supply the water. The irrigation abstraction route, in turn, will demand water from the module that is listed as its upstream module. This upstream module is therefore the supply module.

The irrigation block then solves the soil- and groundwater storages and the return flows that it will generate, based on the assumption that the demands will be met.

When the supply module is solved, it may happen that the demands in the abstraction route cannot be met. When this happens, the maximum flow that can be supplied will be placed in the abstraction route, and the supply module will notify the irrigation block that something went wrong. When the irrigation block receives this message, it recalculates the return flows and the soil moisture states with the supplied flows.

There are two irrigation models that the user can choose from: the classic WRSM/Pitman model and the WQT model. The WQT irrigation model was extracted and adapted from the conservative pollutant model WQT by Dr Chris Herold.

The class uses an instance of the Raingauge class to keep and supply the rainfall data that is used in the calculations. It also uses two instances of the Route class – one as a single inflow route – the irrigation Abstraction Route – and one as the outflow route to store the simulated return flows.

7.5.1 The Irrigation Module Class: class data record

```
type :: Irrig_Type
```

```
! Irrigation submodel external (user defined) number
```

```
INTEGER :: NRREN
```

```
! Irrigation block data file name
```

CHARACTER(len=MNFILC) :: RRFile

! Irrigation module name

CHARACTER(len=20) :: RRNAM

! Whether the Irrigation module is valid

LOGICAL :: VALID

! Whether the irrigation data was changed since it was read from the file

LOGICAL :: CHANGED

! The type of model that this irrigation module must use:

! 1x for WRSM/Pitman or

! 2x for WQT.

! If new versions of the WRSM/Pitman file are written with extra data

! then these will be numbered 11, 12, etc. and those for WQT will be 21, 22, etc.

! There is therefore enough space for further changes in data files since

! we take only the first digit as the model type number and any digits

! after that as the file version number.

! Note that datafiles for the WQT irrigation model are (for now) the same

! as those that are used in the WQT model proper – we simply ignore the

! salt component. There is one additional version label on the first line of the file.

! It would be a good idea to make WQT conform to this scheme as well.

INTEGER :: MODEL

! and the x in the above for the file version

INTEGER :: VERSION

! Number of points to define growth in irrigation area

INTEGER :: NYRR

! Year/Area under irrigation in km² for specified year pairs

type(YearValue_Type) :: ARR(MNRRAGT)

! Monthly crop demands in mm

REAL :: CDEM(12)

```

! Effective rainfall factors

REAL :: EFFP(12)

! Maximum annual allocation in mm

REAL :: PERM

! Proportion of total area under irrigation

REAL :: PINDEX(12)

! MAP over irrigation area in mm

REAL :: RRMAP

! Raingauge reference number

INTEGER :: NPI

! Monthly pan evaporation

REAL :: APAN(12)

! Monthly pan factors.

REAL :: APF(12)

! Abstraction route number (user defined)

INTEGER :: NRRARN

! Return flow route number (user defined)

INTEGER :: NRRRRN

! Area under irrigation

REAL :: AIRR

! Monthly irrigation demands

REAL :: RRMM(12)

! Return route flow (percentage of abstraction)

REAL :: RRPC

! Specific for the WQT model:

! The annual water supply factor

REAL  :: ANNFAC

```

```

! Interpolation type for INTENT (linear, exponential or none) for Area
    INTEGER :: KA

! Maximum water allocation ( MCM per yr )
    REAL    :: RRMA

! Number of data points for water allocation
    INTEGER :: NPMA

! Interpolation type for INTENT (linear, exponential or logarithmic) for
! Water allocation
    INTEGER :: KMA

! Year/Irrigation Water Allocation Growth values in a specific year pairs
type(YearValue_Type) :: WAG(MNRRAGT)

! Abstractions flows file name
    CHARACTER(len=MNFILC) :: FQ

! Related salt washoff submodel number (is rrs w in the original WQT)
    INTEGER :: IRRSW

! Transfer canal – proportion of flow loss
    REAL    :: RRTL PQ

! Transfer canal seepage – proportion of seepage that returns to
! the return flow
    REAL    :: TLSQRF

! Transfer canal – proportion of salt loss (needed to save the
! module data file.)
    REAL    :: RRTLPS

! Irrigation Efficiency factor
    REAL    :: RRIE

! Return flow factor
    REAL    :: RRLF

! Proportion of return flow upper zone

```

REAL :: RRPRFU
! Proportion of return flow lower zone

REAL :: RRPRFL
! Deep percolation salt concentration factor (needed to save the
! module data file.)

REAL :: RRSCF
! Proportion of irrigated land salt loss (needed to save the
! module data file.)

REAL :: RRPSL
! Salt load applied to irrigated land 1 and 2 (needed to save the
! module data file.)

REAL :: RRSLD(1)
! Initial salt load – lower zone (needed to save the
! module data file.)

REAL :: RRSLD(2)
! Initial salt load – upper zone (needed to save the
! module data file.)

REAL :: RRSSUI
! Initial salt load – lower zone (needed to save the
! module data file.)

REAL :: RRSLI
! Soil moisture storage capacity upper(mm.)

REAL :: RRHSU
! Soil moisture storage capacity lower(mm.)

REAL :: RRHSL
! Soil moisture storage target (mm.)

REAL :: RRHT

! Soil moisture storage initial (mm.)
REAL :: RRHI

! Soil moisture storage, effective area, current time period
REAL :: RRHE

! Soil moisture storage, effective area, previous time period
REAL :: RRHEP

! The base allocated irrigation area.
REAL :: RRABAS

! The effective area is the area that can be irrigated with the
! available water.
! The non-effective area is the area for which there is not enough
! water available to irrigate it.
! The effective irrigation area for the current time period
REAL :: RRAE

! The effective irrigation area for the previous time period
REAL :: RRAEP

! The non-effective irrigation area
REAL :: RRANE

! The non-effective irrigation area for the previous time period
REAL :: RRANEP

! Soil moisture storage, non-effective area, current time period
REAL :: RRHNE

! Soil moisture storage, non-effective area, previous time period
REAL :: RRHNEP

! Effective rainfall factor
REAL :: RRERF(12)

! Effective rainfall limit
REAL :: RRERL1

```

! Effective rainfall limit

REAL    :: RRERL2

! RRID is the irrigation demand. This value is used in RR_Demand and
! in RR_Calculate2.

REAL    :: RRID

! APANF is the monthly pan factors – use APF instead

! REAL    :: APANF(12)

! The number of crops for which there are crop factors

INTEGER :: NCPS

! PE is used for potential evaporation in the WQT code – we use APAN instead

! REAL    :: PE(12)

REAL    :: RRTCF(12)

! The Mean monthly lake evaporation (mm.). Calculated in RR_Demand
! and used in RR_Calculate2

REAL    :: RRPEL(12)

! The crop factor for each crop

REAL    :: CPF(MNRRCROPS)

! Total crop demands for 12 months.

REAL    :: RRTCD(12)

! The number of crops factors (one for each type of crop, for each month)

REAL    :: CF(MNRRCROPS,12)

! Number of Year/Return Flow Growth points

INTEGER :: NRFA

! The interpolation option to be used for Irrigation Return Flow Growth

INTEGER :: KRFA

! Year/Irrigation Return Flow Growth values in a specific year pairs

type(YearValue_Type) :: RFG(MNRRAGT)

! Number of Year/Irrigation Efficiency growth points

```

INTEGER :: NIEG

! The interpolation option to be used for Irrigation Efficiency growth

INTEGER :: KIEG

! Year/Irrigation Efficiency values in a specific year pairs

type(YearValue_Type) :: IEG(MNRRAGT)

! The gross abstraction for the irrigation block

REAL :: RRQG

! Indicator to show that we are using a file with input data (used in WQT only)

INTEGER :: RRSFI

! Transfer canal flow loss

REAL :: RRTLSQ

end type Irrig_Type

7.5.2 The Irrigation Module Class: class Variables

MNRRS - Number of Class data records created

NRR - Number of class data records in use

CURRENTFILEVERSION - The file version number that is used when writing data files

7.5.3 The Irrigation Module Class: Methods

7.5.3.1 Public methods

Public methods can be called from any other module or function that contains the statement

use irrigation_module

RR_AddInFlow

Add an inflow route to the specified Irrigation Module – called when the user created a new route with this particular module as the downstream module.

RR_AdjRetFlow

Adjust the return flow in the specified irrigation module. This method is called by a supply module (i.e. the Reservoir or Channel Module that supplied the irrigation abstraction route) when the supply did not meet the demand. Where the irrigation module must use the WQT model, call RR_Calculate2 with the limited inflow message set to true.

RR_AddOutFlow

Add an outflow route to the specified Irrigation Module – called when the user created a new route with this particular module as the upstream module.

RR_Calculate

Run the simulation for the specified irrigation module. Choose the correct simulation algorithm for the irrigation model (WRSM or WQT) that was selected for the specified module.

RR_CanDelete

Call RR_HasChanged to check whether the specified irrigation module data has been changed since it was last saved. Ask whether to save. Call RR_Save if necessary. Return true if the instance can be deleted or false if not.

RR_Check

Check the data in the specified irrigation module and report success if specified, otherwise only report errors.

RR_Delete

Call RR_CanDelete. Delete the specified irrigation module if RR_CanDelete says that it is ok.

RR_DeleteAll

Call RR_Delete for all the Irrigation Module instances in the network.

RR_DelInflow

Delete the Inflow route to the specified irrigation module – called when the user deleted a route with this particular module as the downstream module. Set the module to invalid.

RR_DelOutflow

Delete the Outflow route to the specified irrigation module – called when the user deleted a route with this particular module as the upstream module. Set the module to invalid.

RR_Edit

Edit the data for the specified irrigation module – use the method RR_Edit_GS to clear, set or read the data.

RR_GetName

Return the name of the specified irrigation module.

RR_GetSortedRS

Return a 48 character long string which contains

1. the external irrigation module number
2. the irrigation module name

The argument index is the index number, i.e. the xth runoff module to be found in the sequence of sorted external numbers.

RR_HasChanged

Report whether the data for the specified irrigation module has changed since it was last saved to file.

RR_HowMany

Report how many instances of Irrigation Modules there are (or: how many Irrigation modules are in the Network)

RR_IsValid

Check whether the data in the specified irrigation module has passed the RU_Check test.

RR_NextNewEN

Returns the next (lowest) number for a new irrigation module external number.

RR_Print

Produce a file with the data for the specified irrigation module with explanations and print it.

RR_PrintAll

Call RR_Print for all instances of irrigation modules.

RR_Read

Create a new instance of an irrigation module data record and read the file with the data if such a file was specified.

RR_Reset

Reinitialise all the calculated variables in the specified irrigation module and zeroize the outflows.

RR_Save

Save the data to a module data file.

RR_ShowAll

Show all the instances of irrigation modules and an indicator about their validity on the screen.

7.5.3.2 *Private methods*

Private methods can only be called from functions within the class.

RR_Calculate1

Calculate the abstractions and the return flows for the specified Irrigation module for the specified time period using the WRSM/Pitman model. For use ONLY by RR_Calculate.

RR_Calculate2

Calculate the abstractions and the return flows for the specified Irrigation module for the specified time period using the WQT irrigation model. The message 'limit' determines whether the gross demands are to be calculated, or whether the simulation is to be recalculated as a result of a supply failure.

RR_CanDelete

Check whether the specified Irrigation module can be deleted. An Irrigation block may only be deleted if it has been saved when the data has been changed.

RR_CheckAll(reportok)

Checks all the Irrigation modules in the current network. If all Irrigations seem to be valid, it return 1 otherwise return a 0 The argument reportok is optional. If it is present and 'Y' then every module that is valid will be reported, otherwise correct Irrigations are not reported.

RR_CHSTOR

Process the changes in groundwater zone areas in the specified irrigation module. Only called from within Calculate2 (which was originally called RRQSUB in WQT)

The name of this method was kept so that the original structure of the WQT program could be conserved to some extent.

RR_Demand

Calculate the 12 monthly irrigation demands for the specified irrigation block. Modified from the original WQT routine to calculate demands for one year only.

RR_Edit_GS

Clear, Set or Save the values for the Edit dialog for the specified Irrigation module

Return 1 if things went OK or 0 if there was an error.

RR_EHeader

Creates the top part of an error message. Error messages always start:

Irrigation module xxxxxx (the name of the module)

Module Number: yyy

and then the rest of the message.

So that we don't do this x times, do this as a subroutine.

RR_GetIndex

Gets the index number of the array Irrig_PA for the specified Irrigation module. Return the index number within the array or 0 if it could not be found.

RR_GetVersion

Determine the irrigation model (WRSM or WQT) to which the datafile pertains and the version number of the format of the file.

If the data file is invalid, return imodel as -1 and iverision as -1.

RR_Init

Initialise all the variables in the Irrigation data record of the specified Irrigation module to harmless values. Return 1 if completed successfully or issue warning and return 0 if failed.

RR_New

Assign a new Irrigation data record and return the internal number of that record. If there is no unused record available to assign i.e. NRR equals MNRRS then attempt to assign a new data structure.

RR_PrintM

Prints the data of the specified Irrigation Module to a file that is already opened. For the WRSM model use the method RR_PrintM1 and for the WQT model use the method RR_PrintM2.

RR_PrintM1

Create the printout file for the parameters for a classic WRSM/Pitman irrigation model data file.

This function is only called by RR_PrintM.

RR_PrintM2

Create the printout file for the parameters for a WQT irrigation model data file.

This function is only called by RR_PrintM.

Function RR_Read(nc,kin1)

Read a file that is written in the original format for the specified irrigation module.

Function RR_Read2(nc,kin1)

Read a file that is written in the WQT format for the specified irrigation module.

Function RR_Read3(nc,kin1)

Read a file that is written in the WRSM + WQT format for the specified irrigation module.

RR_RRDIN

Calculate unit irrigation demand (in mm) for the function RR_Calculate2 for the specified irrigation module and for the specified month and time period.

This routine was kept so that the original structure of the WQT program could be conserved to some degree

RR_SetActives(nc)

Sets tabs and fields either active or disabled depending on the model type that the specified irrigation module will use. Returns 1 if successful or 0 if failed.

Note: this function only works on the dialog RR_Edit and its sub-dialogs

RR_Write

Write the data for the WRSM model in the specified Irrigation module to the input file that was used before. If there was no file read, create a new file with the standard name in the input directory.

NOTE: Never call this function directly – there are additional checks that are made in RR_Save which are carried out before this function is called.

Function RR_Write2(mm)

Writes the data for the WQT model in the specified Irrigation module to the input file that was used before. If no file was read, create a new file with the standard name in the input directory.

NOTE: Never call this function directly – there are additional checks that are made in RR_Save which are carried out before this function is called.

7.6 The Reservoir Module Class

The reservoir class is used to model a Reservoir or Dam. The reservoir data record contains data about the volume of storage and area of a real-world reservoir and inflow and outflow routes.

7.6.1 The Reservoir Module Class: class data record

type :: Dam_Type

! Reservoir external node number

INTEGER :: NRVEN

! Reservoir data file name

CHARACTER(len=MNFILC) :: RVFile

! Reservoir name

CHARACTER(len=20) :: RVNAM

! Whether the Reservoir module is valid

LOGICAL :: VALID

! Whether the reservoir data was changed since it was read from the file

LOGICAL :: CHANGED

! Number of route sources


```

INTEGER :: NRTIR

! Inflow route numbers (external)

INTEGER :: JRTIR(MNRVIN)

! Number of release routes

INTEGER :: NRTOR

! Release route numbers (external)

INTEGER :: JRTOR(MNRVOUT)

! Storage state for restricted demands for every one of the release
! routes. If the storage falls below this value, the restriction factor
! (see below) comes into play

REAL :: RVRES(MNRVOUT)

! Restriction factor on demands for every one of the release routes
! just in case the storage state falls below the cutoff reserve RVRES

REAL :: RVRF(MNRVOUT)

! Reservoir spillage route number (external)

INTEGER :: JOUTR

! Current reservoir area

REAL :: ANOW

! No. of years with info. on capacity/area

INTEGER :: NYRV

! The year/capacity array

type(YearValue_Type) :: RVS(MNRVAVT)

! The year/area array

type(YearValue_Type) :: RVA(MNRVAVT)

REAL :: RVMAP

! Reference number of Precipitation file as stored in array FILES.

INTEGER :: NPR

! Power of area/storage relationship

```

```

REAL :: RVPOW

! Current reservoir storage state

REAL :: SNOW(12)

! Storage at end of Previous MonTH (or initial storage)

REAL :: SPMNTH

! Pan evaporation

REAL :: RPAN(12)

! Pan evaporation factors

REAL :: RPF(12)

! Gross reservoir evaporation

REAL :: RVE(12)

! Reservoir spillages

! REAL :: RVSPIL(MNTPS)

! Storage state of reservoir – monthly

REAL :: QRV(MNTPS)

! Whether the initial storage state of the reservoir is specified by the

! user

LOGICAL :: UDEFSTIN

! Initial storage volume at beginning of simulation

REAL :: RVSTIN

! Initial storage area at beginning of simulation

REAL :: RVAINI

! Full supply volume for the year

REAL :: SFULL

! Full supply area for the year

REAL :: AFULL

! Indicator whether the module failed

! To keep whether the module failed – in the first character

```

! we keep whether the defined flow demands on the module failed

! in the second character, we keep whether the calculated flows

! failed.

! Codes:

! 0000 – no failure

! 0001 – minor failure on calculated outflow (< 0.009 per month)

! 0010 – major failure on calculated outflow (> 0.009 per month)

! 0100 – minor failure on defined outflow (< 0.009 per month)

! 1000 – major failure on defined outflow (> 0.009 per month)

! or any combination of these, e.g.

! 2222 – minor and major failure on all route types. Note that

! since a char variable has only 8 bits, the highest

! number of failures can only be 255.

CHARACTER(len=4) :: RVFAIL

end type Dam_Type

7.6.2 The Reservoir Module Class: class Variables

MNRVS - The number of reservoir data records currently created
 NRV - The number of reservoir data record in use

7.6.3 The Reservoir Class: Methods

7.6.3.1 Public methods

Public methods can be called from any other module or function that contains the statement

use reservoir_module

RV_AddInFlow

RV_AddOutFlow

RV_Calculate

RV_CanDelete

RV_Check

RV_Delete

RV_DeleteAll

RV_DelInflow

RV_DelOutflow

RV_Edit

RV_Exists

Report whether a specified reservoir exists by either true or false.

RV_GetDraft

Gets the draft for the specified reservoir for a specific time period (itp) from the data stored.

If the function is successful, it returns the value otherwise return a -999.99 and issue warning messages.

RV_GetName

Return the name of the specified reservoir in a string with a length of 20 characters.

RV_GetMaxCap

Returns the greatest possible Full Supply Volume that the reservoir attains during the simulation period. (Used to determine maximum value of a graph.

RV_GetSpareCapacity

Return the spare capacity, i.e. the capacity that the reservoir is below FSL for the specified reservoir at the beginning of the specified time period.

The method is used to determine the approximate maximum demand for a diversion channel.

The method reports the maximum demand as it is at the end of the previous month and does not take the rainfall that may fall nor the evaporation that occur over the reservoir in the next month, nor that there may be inflows from elsewhere. The method will therefore not estimate the demands in more complex reservoirs correctly.

This is a stopgap method which can only be replaced by a prescient demand centre Reservoir module that can adjust the monthly demand by taking all in- and outflows into account.

RV_GetSortedRS

Returns a 48 character string in which there is

1. the external reservoir number
2. the reservoir name

for the xth reservoir to be found in the sorted sequence of reservoirs.

RV_GetValue

Return the storage value for the specified reservoir (external number) for the specific time period (itp) from the data stored. Return the value if the function is successful otherwise return -999.99 and issue warning messages (if invalid reservoir number or invalid time period).

RV_GetYearBuilt

Return the year in which the specified reservoir was built.

RV_HasChanged

Check whether the data of the specified reservoir was changed since the data was loaded from the data file or since it was last saved.

RV_HowMany

Report how many reservoirs there are in the current Network.

RV_IsValid

Check whether the data of the specified reservoir is valid – i.e. no has missing or wrong data.

RV_NextNewEN

Get the next new external number that can be used as a unique number for a reservoir.

RV_Print

Print the specified module using the method RV_PrintM

RV_PrintAll

Call RV_Print for all reservoir modules that are currently in the Network by calling RV_Print for every one of them.

RV_Read

Read the parameter data file associated with the specified Reservoir. If necessary create a new Reservoir Data Record. The specified file will be read. The third message (iread) is optional. If only 2 arguments are passed, i.e. module number (mm) and a file name (FILEN) then:

1. A record for the Reservoir module mm will be created if possible.
2. The file is read.

When FILEN is blank, create a standard file path from the system name, RV, the number mm and the data directory before attempting to read this file.

If iread is present but greater than 0 then the file will be read or a file name will be created and read.

If iread is present but 0, the file will not be read, even if one is supplied. A record will be created and the default standard file path created but there will be no other data in the record.

RV_Reset

Reset the storages and other parameters in preparation of a simulation run.

RV_Save

Save the parameters of the specified reservoir module – check whether it is OK to overwrite the file, open the file, then use RU_Write to write the data.

RV_SaveStorage

Store the storage state of (a) reservoir(s) in an answer file, using the private function RV_WriteStor.

RV_ShowAll

Show all Reservoir Modules that are currently in the Network.

RV_WriteSErr

Write the simulation errors (if there are any) to the standard simulation error file in the EString class

RV_WriteSummary

Write the summary elements (storages) to the standard summary file in the Estring class.

7.6.3.2 Private methods

Private methods can only be called from functions within the class.

RV_CheckAll(reportok)

Check all the Reservoirs data records current in the Network. If all Reservoirs seem valid, return 1 otherwise return 0. The message reportok is optional. If it is present and 'Y' report every module that is valid. If reportok is 'N' do not report Reservoirs that are correct, but report those that have errors.

RV_Edit_GS

Clear, set or save the values in the Edit dialog for the reservoir specified by the internal number.

Options:

- 0 – clear the fields
- 1 – set the values from what is currently in memory
- 2 – save the values from the dialog to the memory.

Returns 1 if things went OK or 0 if an error occurred.

RV_EHeader

Create the top part of an error message in the EString class for the specified Reservoir.

Error messages always start:

Reservoir module xxxxxx (the name of the module)

Module Number: yyy

RV_GetIndex

Report the index number of the array Dam_PA where the reservoir with the specified external number is stored. Return the number of the index, or 0 if the record could not be found.

RV_GetSortedRS

Returns a 48 character long string in which there is

1. the external reservoir number
2. the reservoir name

for the specified Reservoir. The message is the index number, i.e. the xth reservoir to be found in the sorted sequence.

RV_Init

Initialise all the variables in the Reservoir data record specified by its internal record number, return 1 if completed successfully or 0 if failed.

RV_New

Assign a new Reservoir data record and return the internal number of the record. If there is no record available to assign i.e. NRV equals MNRVS then attempt to assign 3 new data records, initialise them with RV_Init.

RV_PrintM

Print the parameter data of the specified Reservoir Module with external number mm to the file that is already opened as a specified unit.

RV_WriteSErr

Write the simulation error summary to the file that is opened as the standard error file EF in the eststring module.

RV_WriteStor

Writes the storages time series for the specified Reservoir Module to the specified file.

7.7 The Channel Module Class.

The Channel Module Class (also called the Channel Reach Module class) is the class that is used to model an active stretch of a river channel. Routes are not river channels because routes are lossless and one cannot add inflow or outflow routes to a route.

Apart from inflow and outflow routes, Channel Module classes may also have special features such as wetlands and diversion channels.

7.7.1 The Channel Module Class: class data record

type :: Channel_Type

! External module number.

INTEGER :: NCREN

! Channel Reach data file name

CHARACTER(len=MNFILC) :: CRFile

! Indicator to show that the data for this module

! has been changed since it was read from the file.

LOGICAL :: CHANGED

! Indicator to show that the data for this module is valid

! If all data has been checked, VALID will be set to TRUE

LOGICAL :: VALID

! Node name

CHARACTER(len=20) :: CRNAM

! The MAP of the wetlands

REAL :: CRMAP

! Monthly bed loss from channel

REAL :: CHBL

! Wetland/Aquifer storage ($m^3 \cdot 10^6$)

REAL :: CHC1

! Wetland/Aquifer area (km^2)

REAL :: CHC2

! Wetland/Aquifer recharge coefficient

REAL :: CHC3

! Gross wetlands evaporation

REAL :: CHE(12)


```

! Route no. for MAIN or PRIMARY outflow from the channel

INTEGER :: JOUTC

! Inflow route numbers (user defined)

INTEGER :: JRTIC(MNCRIN)

! Outflow route numbers (user defined)

INTEGER :: JRTOC(MNCROUT)

! Number of inflow routes

INTEGER :: NRTIC

! Number of outflow routes

INTEGER :: NRTOC

! Rainfall file name

! CHARACTER(len=MNFILC) :: RainFile

! Raingauge number

INTEGER :: NPC

! Monthly Pan evaporation

REAL :: WPAN(12)

! Monthly pan factors

REAL :: WPF(12)

! Current Wetland/Aquifer storage

REAL :: WETS

! Whether the module failed

! In the first element we keep whether the defined

! flow demands on the module failed.

! In the second element, we keep whether the calculated flows

! failed.

! Codes:

! 0000 – no failure

! 0001 – minor failure on calculated outflow (< 0.009 per month)

```

! 0010 – major failure on calculated outflow (> 0.009 per month)
! 0100 – minor failure on defined outflow (< 0.009 per month)
! 1000 – major failure on defined outflow (> 0.009 per month)
! or any combination of these, e.g.
! 2222 – minor and major failure on all route types. Note that
! only the first 255 errors of any type will be listed
! in the error file.
!

INTEGER :: CRFAIL(4)

! Whether the channel reach has wetlands 1 data

LOGICAL :: HASWETLANDS1

! Whether the channel reach has wetlands 2 data

LOGICAL :: HASWETLANDS2

! Whether the channel reach has diversion route data

LOGICAL :: HASDIVERSION

! The type of wetlands model to use:

! 0 for none, 1 for Basic, 2 for Comprehensive, 3 for Diversion

INTEGER :: WLMODEL

! Specifically for the Comprehensive Wetlands model:

! Area of wetland at bankfull level of channel (km²)

REAL :: AWB

! Volume of wetland at bankfull level (mcm)

REAL :: VWB

! Power of area-volume relationship

REAL :: PAV

! Bankfull capacity of river channel (mcm/month)

REAL :: QBNK

! Diversion capacity if off-channel storage scheme (mcm/month)

```

REAL :: QDIV

! Proportion of flow in excess of bankfill into wetland

REAL :: PRQIN

! Proportion of wetland volume (above bankfill) into channel

REAL :: PRVOUT

! Wetland local inflow route number (USER DEFINED)

INTEGER :: WRTIN

! Wetland local outflow route number (USER DEFINED)

INTEGER :: WRTON

! Comprehensive wetland model, area of wetland

REAL  :: AWET3

! Comprehensive wetland model, volume of wetland

REAL  :: VWET3

! The constant in the wetland area-volume equation

REAL  :: WC

! Specifically for the diversion route

! Bankfull capacity of river channel (mcm/month)

REAL :: DRQBNK

! Diversion capacity if off-channel storage scheme (mcm/month)

REAL :: DRQDIV

! Diversion efficiency

REAL :: DRPRQIN

! The Diversion Route Outflow Route Number (User Defined)

INTEGER :: DRTON

end type Channel_Type

```

7.7.2 The Channel Module Class: class Variables

MNCRS - The number of Channel Module Data Records that have been created

NCR - The number of Channel Module Date Records that are currently in use

CURRENTFILEVERSION - The file version to use when writing files.

7.7.3 The Channel Module Class: Methods

7.7.3.1 *Public methods*

Public methods can be called from any other module or function that contains the statement

use channel_module

CR_AddInFlow

Add the specified inflow route to the specified Channel Module.

CR_AddOutFlow

Add the specified outflow route to the specified Channel Module

CR_Calculate

Run the simulation for the specified Channel Module for the specified time period. If the specified Channel Module has special features, also call CalcWetlands1, CalcWetlands2 or CalcDiversion.

CR_CanDelete

Check whether the specified channel reach can be deleted. A channel reach may only be deleted if it has been saved if the data has been changed. Call CR_HasChanged.

CR_Check

Check the data of the specified Channel reach and report if there are errors. If called with the reportok message set to 'Y', report if the data for the specified class is OK

CR_Delete

Destructor function of a specified Channel Reach. Delete all data to do with the specified Channel Reach. It return 1 if the deletion was successful, or 0 when it was not.

CR_DeleteAll

Delete all channel reaches in the Network by calling CR_Delete for each instance.

CR_DelInflow

Remove the specified inflow route from the specified channel reach. Note that this does not remove the route itself – it is called when the route is removed.

CR_DelOutflow

Remove the specified outflow route from the specified channel reach. This does not remove the route, but only the reference to it. The method is called when the route is removed.

CR_Edit

Edit the specified Channel Module. Use CR_SetActives to influence the dialog and CR_Edit_GS to set and get the values.

CR_GetName

Return the name of the channel reach.

CR_GetSortedRS

Returns a 48 character string that contains

1. the external channel module number
2. the channel module name

for the specified index number, (i.e. the xth channel module to be found) in the sequence of sorted channel reaches.

CR_HasChanged

Determine whether the data of the specified Channel Reach has been changed since it was read or since it was last saved.

CR_HowMany

Determine how many Channel Modules there are in the current Network

CR_IsValid

Report whether the data for the specified Channel Module is valid.

CR_NextNewEN

Determine the next new unique module number that can be used.

CR_Print

Print the data parameters for the specified Channel Module, using PrintM

CR_PrintAll

Print the data parameters for all Channel Modules, using CR_Print

CR_Read

Reserve a new Channel Module data record. Use CR_New. If no data records are available, CR_New will create and initialise it. Read the data file – if one was specified. If no file was specified, initialise the record to harmless values, and await editing later.

CR_Reset

Reset the data for the specified Channel Reach prior to a (re-) run of the simulation

CR_Save

Save the data for the specified Channel Module to the file. Checks on- and opening of files only – use CR_Write to do the actual writing.

CR_ShowAll

Display all the Channel Modules on the screen, together with an indicator to show whether they are valid or not.

CR_WriteSErr

Write the simulation errors to the simulation error file in the Estring class.

7.7.3.2 Private methods

Private methods can only be called from functions within the class.

Function CR_New()

Assign a new channel data record and return the internal number of the record. If there is no free record available to assign i.e. NCR equals MNCRS then attempt to assign a new data record.

CR_Init

Initialises all the variables in the specified Channel reach record. Return 1 if successful or 0 if something went wrong.

CR_CheckAll

Check all the channel modules that are currently in the network. If all channel reaches seem to be valid, it returns 1 otherwise return a 0. The message reportok is optional. If it is present and 'Y' then every module that is valid will be reported, otherwise correct channel reaches are not reported.

CR_Write

Write the data of the specified Channel Reach the input file that was used before. If there was no file read, then a new file with the standard name will be created in the input directory.

NOTE: Never call this function directly – there are additional checks that are made in CR_Save which are carried out before this function is called.

CR_GetIndex

Retrieve the index number of the array Channel_PA where the channel module with the specified external number can be found. Return the number of the index, or 0 if it could not be found.

CR_Edit_GS

Clear, set or save the values for the specified channel module in/from the Edit dialog.

Returns 1 if things went OK or 0 if there was an error.

CR_CalcWetlands1

Calculate and set the outflows that are generated in the main outflow route of the specified channel module for the specified time period. This is the routine for the 'Basic' Wetlands model.

Note: this method adjusts the outflow in the main outflow route only.

CR_EFHeader

Set the header for an error file message in the string in the EString module so that it can be written with the function ES_WriteEF to the error file.

CR_CalcWetlands2

Calculate the losses as a result of wetlands by means of the COMPREHENSIVE wetlands algorithm for the specified channel module and time period.

Once completed, adjust the outflows in the main outflow route of the specified channel reach module.

Called from the CR_Calculate function. The function returns 1 if the function executed OK or 0 if something went wrong. Note: this method adjusts the outflow in the main outflow route and in the local outflow / abstraction route (if present).

Function CR_CalcDiversion(nc, ntp)

Calculate the flows in a diversion route for the specified Channel Module in the specified time period. Once completed, adjust the outflows in the main outflow route of the specified channel reach. Called only from the CR_Calculate method.

Return 1 if the function executed OK or 0 if something went wrong.

Note: this method adjusts the outflow in the main outflow route and sets the flows in the diversion route.

CR_EHeader

Create the top part of an error message. Error messages always start as:

Channel module xxxxxx (the name of the module) Module Number: yyy

and then the rest of the message.

CR_WriteSErr()

Write to the simulation error summary to the file if the file is opened. The simulation error file is kept in the class ESTRING in the estring_module.

CR_PrintAll()

Print all the channel modules in the system.

CR_PrintM(mm, KOUT1)

Print the data of the specified Channel Module to the file opened as a Fortran unit.

CR_GetVersion(nr,fdnam)

Obtain the version number of the specified file and make sure that it contains the data of the specified Channel Module. Return the version number of the file, or 0 if no version number was found in the file. Since 0 is the first version number, this is OK. If the file has no data (or the data is wrong), issue a warning and return -1

CR_SetActives

Set tabs and fields in the Edit dialog to either active or disabled depending on which one of the wetland model types is in use for the specified module. Return 1 if successful or 0 if failed.

Method only works on the dialog CR_Edit and its sub-dialogs.

8 THE CLASSES FOR STATISTICS AND PLOTTING

In this section we discuss the classes that are used to calculate the flow statistics of a route and also the class that is used to plot the flows in routes and the storages in reservoirs.

8.1 The Stats Class

This class is a single instance class that is used to calculate and display the flows statistics of a route. If observed flows are present, the statistics for both the observed and simulated flows are calculated.

8.2 The Stats Class: class data record

```
type :: Stats_Type

! Mean monthly observed flows as percentage of Mean Annual Runoff (MAR)

REAL  :: POBS(12)

! Mean monthly simulated flows as percentage of Mean Annual Runoff (MAR)

REAL  :: PSIM(12)

! Mean log of observed flows

REAL  :: QMLOBS

! Mean log of simulated flows

REAL  :: QMLSIM

! MAR of observed flows

REAL  :: QMOBS

! MAR of simulated flows

REAL  :: QMSIM

! Std. deviation of logs of observed flows

REAL  :: SDLOBS

! Std. deviation of logs of simulated flows

REAL  :: SDLSIM

! Standard deviation of observed flows

REAL  :: SDOBS

! Standard deviation of simulated flows

REAL  :: SDSIM
```

! Seasonal index of observed flows
REAL :: SIOBS

! Seasonal index of simulated flows
REAL :: SISIM

! Whether the Simulate calculations were done
LOGICAL :: SIMDONE

! Whether the Observed calculations were done
LOGICAL :: OBSDONE

! Coefficient of variability – simulated flows
REAL :: CVSIM

! Coefficient of variability – observed flows
REAL :: CVOBS

! Autocorrelation coefficient of observed flows
REAL :: ACOBS

! Autocorrelation coefficient of simulated flows
REAL :: ACSIM

! Coefficient of skewness of observed flows
REAL :: CSOBS

! Coefficient of skewness of simulated flows
REAL :: CSSIM

! Range of observed flows
REAL :: RGOBS

! Range of simulated flows
REAL :: RGSIM

! Whether the Statx routine was done for Simulated values
LOGICAL :: STATXSIM

! Whether the Statx routine was done for Observed values
LOGICAL :: STATXOBS

! Whether the edit screen must be updated

LOGICAL :: CALCULATED

end type Stats_Type

8.3 The Stats Class: class Variables

This class only features a single instance of the class data record called

ST_Data

8.3.1 The Stats Class: Methods

8.3.1.1 Public methods

Public methods can be called from any other module or function that contains the statement

use stats_module

Result

Save the observed and simulated statistics for the specified route (specified as the EXTERNAL route number) for the specified time period (start and end year) to the file that is already opened at the specified Fortran unit KOUT. An optional message determines whether the Runoff Module parameter data must be written to the file or not. Uses STATX.

ST_Route

Display the dialog box IDD_STATISTICS to allow the user to select a route and to displays the statistics once they are calculated.

ST_SaveAll

Save all the routes for which observed and simulated statistics can be produced to a file. If the optional message mprint is TRUE, print the file.

8.3.1.2 Private methods

Private methods can only be called from functions within the class.

COMPAR

Displays a comparison of the statistics between the observed and simulated values in the specified route for the specified time period and give hints on how to improve the fit.

The method uses the values in the ST_Data record, and should therefore only be called after STCALC was called both the observed and the simulated values. The method will write to the dialog IDD_STATISTICS.

STATS

Calculates the statistics of the observed and simulated flows in the specified route for the time period specified as a start and end year. The route number is the user defined route. Check the

feasibility of the route and time data and call STCALC for the simulated flows. Check whether Observed data is available for the route in the class OD_Data, load it and call STCALC for the observed data.

STATX

Calculate additional flow statistics for the specified (user defined) route number starting at a specified time period and ending a specified number of years later. The message icode determines whether to use observed or simulated data for the statistics. Calculate coefficient of variability,

coefficient of skewness, and mean monthly flow, range, autocorrelation coefficient, save them in the data record and set the indicators to show that the statx routine has been executed.

STCALC

Calculate the statistics for the specified route for the specified start time period and the number of years. Differentiate between observed and simulated flows (specified as a message in the call).

If simulated flows are specified, retrieve the data from the Route class.

If observed flow is specified, make sure that the correct data is loaded in the observed data array in the class OD_Data. Use the data from OD_Data to generate Observed data statistics.

Write the results to record ST_Data.

ST_DialogClear

Clear the dialog IDD_STATISTICS of calculated values.

ST_Reset

Reset all the values for all the data in the class record ST_Data.

9 EDITING, COMPILING AND DEBUGGING

Two versions of Lahey and Winteracter exist, namely version 2 and the latest (July 2006) version 7.1. The Windows version was written with version 2 and the enhanced WRSM/Pitman model which was released in May 2006 at a course will remain in version 2. Version 7.1 was bought with the view of bringing the software up to date and also to make use of the debugger which was not part of version 2. The debugger allows the user to set breakpoints anywhere in the code, to run the model to that point and then single step through program execution examining the values of variables. A great deal of discussion and trial and error analysis took place with the Lahey support developers, Grant Nyland (a computer consultant) and Allan Bailey before being able to compile and debug. Initially the debugger did not work at all with modules but after proving this to Lahey, a patch was received which corrected the bug. Some minor adjustments had to be made in the version 7.1 code to get WRSM/Pitman to compile and run successfully. It appears that the version 2 compiler was not as "strict" as the version 7.1 compiler. Version 7.1 is to be used for the WR2005 and WR2012 projects and will have databases, the GIS Viewer and other enhanced features added.

In version 2, both editing and compiling is done using the Winteracter software (using the WINT icon which one should set up on the desktop). The folder required is C:\WINT\Projects\WRSM2000. There is one exception to this and that is for designing the screen for the various pull-down menus in the model. This is done by means of the WINT dialoged software. Which will be described more later. WRSM/Pitman consists of the following modules that must be compiled in the sequence in which they appear below:

Resource.F90
wrsm2000.F90
TiGenLib.f90
TiEString.f90
wrsm2012.f90
wrsm2010.f90
wrsm2008.f90
wrsm2006.f90
wrsm2004.f90
wrsm2007.f90
wrsm2005.f90
wrsm2003.f90
wrsm2016.f90
wrsm2013.f90
wrsm2015.f90
wrsm2014.f90
wrsm2002.f90
wrsm2001.f90
resource.rc

In addition, the following ".f90" datafiles are required for compilation:

tiestring.f90
tigenlib.f90
lglib.f90

Note: There are now four WINT setups as follows:

- Original WINT system as for the WR2005 study;
- Daily time step system which is WINT_daily;
- During the course of the WR2012 study, enhancements were added to the WINT_New system and

- The final version from WR2012 which contains extensive graphical enhancements by Mr Grant Nyland is WINT_graph.

The WINT_graph system is the one which will be used now and in the future for the monthly time step. The WINT_daily is the system to be used for the daily time step. It does not have the graph enhancements, however, there are only two graphs which are relevant. Although the daily time step version includes the monthly time step version, it should only be used for the daily time step. Any references to WINT apply equally to the other WINT systems.

In version 7.1, editing is done by means of the Lahey Fujitsu system. This is found in Start\Programs\Microsoft Visual Studio .NET 2003 folder. Pitman2005 should be loaded which gives access to all the modules. Compiling is done using the DOS command prompt and the batchfile MAKEWRSM2000 from c:\Pitman2005\source. This batchfile does the compiling and linking. The batchfile is given below:

```
@echo off

cls

SET LFor=RESOURCE.F90 wrsm2000.f90 TiGenLib.f90 TiEString.f90 wrsm2012.f90 wrsm2010.f90
wrsm2008.f90 wrsm2006.f90 wrsm2004.f90 wrsm2007.f90 wrsm2005.f90 wrsm2003.f90 wrsm2016.F90
wrsm2013.f90 wrsm2015.f90 wrsm2014.f90 wrsm2002.f90 wrsm2001.f90 LGLib.f90 windows.f90

SET LObj=RESOURCE.obj wrsm2000.obj TiGenLib.obj TiEString.obj wrsm2012.obj wrsm2010.obj
wrsm2008.obj wrsm2006.obj wrsm2004.obj wrsm2007.obj wrsm2005.obj wrsm2003.obj wrsm2016.obj
wrsm2013.obj wrsm2015.obj wrsm2014.obj wrsm2002.obj wrsm2001.obj LGLib.obj windows.obj

SET LLib= -lib "C:\Program Files\WINT\lib.I9m\winter" -lib comdlg32 -lib winmm -lib winspool -lib shell32 -
lib opengl32 -lib glu32

SET LLibPath="C:\Program Files\Lahey-Fujitsu Fortran\v7.1\Win32\Lib"

SET LInclude="C:\Program Files\WINT\include"

SET LSwm=1065,1603,1636,2005,8692,1636,2005,3100,3111,3117,3125,3205,8692,1636,2005,8692

SET LApp=WRSM2000.exe

echo.

echo Compile WRSM/Pitman using LF95.

echo.

@echo on

LF95 %LFor% -g -ml winapi -c -win -mod .;"C:\Program Files\WINT\lib.I9m" -libpath %LLibPath% %LLib% -
swm %LSwm%

rc /i %LInclude% resource.rc

res2obj resource.res resource.obj

LF95 %LObj% -g -ml winapi -win -out %LApp% -libpath %LLibPath% %LLib%
```

pause

This batchfile sets up debug files (".fwd") which will enable the user to jump from module to module. The Lahey Fujitsu debugger is invoked also in the DOS command line by typing "WINFDB WRSM2000.EXE". This followed by the F8 key will take the user to the first line in the main program module WRSM2001.F90. If the user wants to say examine code in WRSM2004, the File menu should be accessed and WRSM2004 opened. Clicking on the appropriate line will set up a red flag icon Pressing the F8 key (Go) will run the model up to that line where it will break and show the "blue finger" icon. If one wants to single step from here it is done using either the F10 key or the F11 key. The F11 key jumps over "Call" statements to subroutines or functions whereas the F10 key will jump into these subroutines and functions in whichever module they may be. To examine the value of variables, clicking on "Debug" and then "Watch" will enable the user to select a variable and "Add" it to the "Watch window". The value of the variable will then be displayed. The user can Restart the debug session by pressing Restart twice and F8 twice.

The WRSM/Pitman modules are described below :

9.1 Resource.F90

This module is basically set up by "resource.rc" i.e. when "resource.rc" is compiled. The compiler puts all the object IDs that it finds in "resource.f90" into the file "resource.f90". The datafile looks as follows :

```
! Winteracter module created : 03/Oct/2005 12:20:41
```

```
!
```

```
MODULE RESOURCE
```

```
  IMPLICIT NONE
```

```
  INTEGER, PARAMETER :: IDD_ADD_OBS      = 101
```

```
  INTEGER, PARAMETER :: IDD_ABOUT       = 102
```

etc.

There follows a line for every screen or window that appears when you run WRSM/Pitman. The IDD_ABOUT for example is the screen that gives details about the WRSM/Pitman model version, the project management and custodian team (Royal HaskoningDHV), the original programmer's organisation (TiSD), etc. It is given the number 102 automatically when compiled. It is therefore imperative at this stage to deal with the "resource.rc" datafile.

9.2 Resource.RC

This datafile is created and edited using the WINTdialoged software. On opening this file (normally C:\WINT\projects\WRSM2000\resource.rc), the first screen or window will appear (which is the About Window described in 9.1 above). One can then edit or add to this screen/window or dialog as it is technically referred to as. If one wants to choose another dialog, this is done by means of Dialog|Choose which displays the title for the dialog e.g. IDD_ABOUT. All sorts of features can be added such as labels, pictures or frames, group boxes, string fields, integer/real/double precision fields, push buttons, grid fields (tables basically where the user has to enter data), check boxes, radio buttons (to offer alternatives between different options, menu field, progress bars, trackbar, tab control and a select field alignment toolbar. With these features the user can design any type of screen imaginable. The size of dialog can be changed by the user. Dialogs can be added or deleted or modified.

The "resource.rc" datafile should be compiled first as it sets up "resource.f90" which some other modules need.

Full details are as follows:

- go into WINT DIALOG EDITOR;
- open the resource.rc file;
- choose the correct dialog from the available dialog list;
- to change units for example, right click on the label to be changed and change accordingly. Labels can be copied and pasted. Window sizes can be extended;
- save the file and
- go into WINT and compile resource.rc or compile the entire project.

9.3 TiGenLib

This is a library of functions to perform all sorts of general tasks such as string manipulation, define year-value structures, validity of datafiles, etc.

9.4 TiEString

This module deals with all the external strings., particularly error strings.

9.5 WRSM2012.F90

The system global module. This module contains the network data that should be more globally available, such as the data and result directories, the years to start and end the simulation, etc.

9.6 WRSM2010.F90

This file contains all functions and routines that deal with raingauges i.e. the procedure involved to set up a catchment based rainfall datafile from a series of station rainfall datafiles.

9.7 WRSM2008.F90

This file contains all the functions and routines that deal with observation points, also called Gauging Stations.

9.8 WRSM2006.F90

This file contains all the functions concerned with ROUTES.

9.9 WRSM2004.F90

This module governs everything to do with runoff modules and is one of the largest and most important modules. The following functions are included :

Portions of this module are given below and discussed in detail :

The following section is merely a comment to describe anything particular to the module. (given in italics)

```
!
=====
! File WRSM2004.f90
! This module governs everything to do with runoff modules.
! Copyright (c) JP Kakebeeke 2000
```


! The algorithm for RU_Calculate: Copyright (c) Dr W.V. Pitman.
! The algorithms for RU_HGCalcs and RU_HGMoist: Copyright (c) Prof. D. Hughes
! The algorithms for RU_SGCalcs : Copyright (c) K. Sami
! No portion of this code may be reproduced, used as part of other software
! or modified without express permission of the authors.
!
! AKB Modified for new afforestation by AK Bailey 20/02/2001 to 26/03/2001
! AKB2 Modified poor code in imported afforestation and alien vegetation routines
! and added checks to calibration parameters by AK Bailey 19/02/2003
!

=====

There follows the title of the module for use by other modules

module runoff_module

There follows a series of "Use" statements i.e. other modules that are used within this module

use YVtypes_module

use constants_module

use types_module

There follows "implicit none" which means that parameters are not defined as integers, reals, etc. by their first letter

implicit none

!

There follows "public" declarations which means that any module can use these functions

public :: RU_AddInFlow, RU_AddOutFlow, RU_Calculate, RU_CanDelete

public :: RU_Check, RU_CheckAll, RU_CopyClimate, RU_CopyParameters

public :: RU_Delete, RU_DeleteAll, RU_DelInflow, RU_DelOutflow,

etc

!

There follows "private" declarations which means that no any module can use these functions. These are defined as either integers, reals, logical (variables that are either true or false) and arrays. Parameters are also assigned default values at this stage. Arrays are defined as for example C(12) which means that C is an array structure consisting of 12 values. Pointer variables (e.g. see Runoff_PTR below) are also defined which are required to dynamically create and destroy memory required by arrays in particular i.e. instead of defining an array of 1000 values in case they are needed, and using up a lot of memory, only those that are needed are added and this is changed dynamically.

*! For data encapsulation, make everything else private so that it
! cannot be seen by the other modules.*

private

!

INTEGER :: MNRUS = 0

INTEGER :: NRU = 0

! The file version to use when writing files.

! Change this only if you need to save extra parameters, other

```

! than those that are saved already.
INTEGER, PARAMETER :: CURRENTFILEVERSION = 7
LOGICAL :: TRACE = .FALSE.
INTEGER, PARAMETER :: NGWMODELS = 3
! The highest forest algorithm
INTEGER, PARAMETER :: MNFORESTALG = 4
! The highest vegetation algorithm
INTEGER, PARAMETER :: MNVEGALG = 2
!
=====
! The Runoff module data structure
! This structure contains all data to do with a Runoff module.
type :: Runoff_Type
! Runoff submodel number (external)
INTEGER :: NRUEN
! Runoff module data file name
CHARACTER(LEN=MNFILC) :: RUFile
! Indicator to show that the data for this module
! has been changed since it was read from the file.
! AKB AFF check file, next line
! Not needed, commented out, JPK
! CHARACTER(LEN=MNFILC) :: AFFCHKFile
LOGICAL :: CHANGED
! Indicator to show that the data for this module is valid
! If all data has been checked, VALID will be set to TRUE
LOGICAL :: VALID
! Runoff submodel name
CHARACTER(LEN=20) :: RUNAM
! Number of runoff exit routes

type :: Runoff_Pointer
type(Runoff_Type), pointer :: Runoff_PTR
end type Runoff_Pointer

! Whenever a new instance of the Runoff module data structure is made, it is
! first assigned to a temporary runoff module pointer.
type(Runoff_Type), pointer :: Temp_Runoff_PTR

! The array with all the pointers to the runoff structures is called
! Runoff_PA. So that the array may grow, we also have an allocatable
! array called Temp_Runoff_PA
type(Runoff_Pointer), dimension(:), allocatable :: Runoff_PA, Temp_Runoff_PA

```

9.10 WRSM2007.F90

This module contains all the functions concerned with irrigation modules.

9.11 WRSM2005.F90

This module contains all the functions concerned with reservoir modules.

9.12 WRSM2003.F90

This module contains all the functions concerned with channel modules.

9.13 WRSM2016.F90

This module contains all the functions concerned with mining modules.

9.14 WRSM2013.F90

This module contains all the functions concerned with calibration statistics.

9.15 WRSM2015.F90

This module contains all the functions concerned with creation of rainfall files

9.16 WRSM2014.F90

This module contains all the functions concerned with drawing of graphs. For information on adding new graph refer to Special Note 10.7 as this involves changing several modules.

9.17 WRSM2002.F90

This system module contains all the data to do with a network, such as the modules in the network, the network code, the directories, etc.

9.18 WRSM2001.F90

This is the main Windows program. This subroutine processes the menu and sub-menu selections such as File, Edit, View, Run and Plot and their sub-functions.

9.19 .Lib and .mod files

There are a number of “.Lib” and “.mod” files which are automatically set up when the modules are compiled. These files are controlled by the modtable.txt datafile which contains the following:

```
# This file contains Lahey LF90 module information
# Created on Tue Jan 31 14:47:17 2006
# Each line contains the module name, followed by the DOS file
# name used to store module information and library object
# data files, followed by the name of the sourcefile for the module.
# Lines beginning with # are comments.

Winteracter  wintera0 winter.f90

resource    resourc0  C:\WINT\PROJECTS\WRSM2000\RESOURCE.F90

constants_module  constan0 C:\WINT\PROJECTS\WRSM2000\WRSM2000.F90
```

```
globals_module  globals0  :\WINT\PROJECTS\WRSM2000\WRSM2000.F90
runoff_module   runoff_0   c:\wint\projects\WRSM2000
etc.
```

which consist of all the “.lib's” (22 in total) such as runoff_0.lib

and then in WRSM2004.f90 for example, we have :

```
module runoff_module
  use YVtypes_module
  constants_module
  types_module
```

then you have :

```
YVtypes0.mod
constan0.mod
types_m0.mod
```

There are 32 “.mod's” in total.

9.20 Design of Delphi Part of the Code of WRSM/Pitman

There are two Delphi components to the WRSM/Pitman application as follows:

- the main component is **WRSM2000DB.DLL** which is a library in which the database connectivity is implemented as well as graph functions. This library is discussed in detail below, and
- the other component is **WREng.dll** which is a resource library. This library has only one purpose which is to compile the resource file WREng.rc into a DLL so that the application can access the string tables. The only contents of this library are string captions used in the application. This DLL requires no further explanation.

Delphi installation

For this reason, Delphi has to be loaded on the C drive. RHDHV purchased the Delphi system which is the Delphi 2007 for Win32 and contains the serial number C4KN-LRDNW8-3EA5PL-22MD which is required if the system needs to be re-installed. It appears that only Disc 1 of the 2 disks is required. It installs onto the C:\Progam Files (x86)\Codegear\RAD Studio folder. The graph enhancements require a system called TeeChart. Mr Grant Nyland used the Delphi version 7 which is a 2001 version of Tchart and this version is required to be built in when compiling the Delphi code.

9.20.1 Overview of Delphi-Fortran Connectivity Strategy

The main application **WRSM2000.EXE** is a Fortran application compiled with Lahey Fortran 90. The Delphi functionality resides in a DLL called **WRSM2000DB.dll** which is compiled in Delphi 7. A set of functions have been declared and exported in the Delphi DLL using the “**stdcall**” calling

convention. These functions are then linked statically to the Fortran executable with the “[DLL_IMPORT](#)” statement.

9.20.1.1 Linking Strategy and Syntax

Declaration in Delphi:

```
procedure SaveToSptsm(...); stdcall;
```

Export from Delphi:

```
exports  
  SaveToSptsm;
```

Import into Fortran:

```
-implib WRSM2000DB.dll  
-import SaveToSptsm
```

Declaration in Fortran:

```
DLL\_IMPORT SaveToSptsm  
CALL SaveToSptsm(...)
```

The signature of the subroutine is inferred by its first use hence the need for explicit casting of some subroutine parameters.

9.20.1.2 Subroutine Parameters

On the problematic issue of parameters being passed to subroutines, the convention was adopted that a reference to a variable was always passed and passing the variable by value was avoided. The reason for this convention was that it was difficult to predict the width (in bytes) of the passed parameter from the Fortran side when the variable was passed by value. Functions were avoided completely.

Passing by reference in Fortran always results in a four byte wide pointer type being passed for all types of variables. It was sometimes necessary to explicitly cast a variable reference as a pointer type to force Fortran to pass by reference. This was required because an explicit function signature was not declared in the Fortran and the compiler sometimes incorrectly assumed that the variable must be passed by reference. The mechanism to explicitly declare function signatures in LF90 was problematic and was therefore avoided.

A convention was also adopted to only pass native variable types like integers, floats, characters, etc.

Use “var” in Delphi: (var AFeatureCode: integer; var AQuatName: PChar;

- Notice the double dereference for “var ... PChar” – this is necessary.

Use “casting” in Fortran: [CALL](#) SaveToSptsm(LFeatureCode, [POINTER](#)(LQuatName),

Note that integers do not need casting but characters arrays do.

9.20.2 Overview of Database Strategy

A Microsoft Access database was used to implement the WRSM/Pitman database. The database must be in the same location as WRSM/Pitman and must be called WRSM2000.mdb.

There are only two Delphi units containing all the Delphi source code that implements the low level connection to the database – “UFortranDatabaseInterface” and “UAccessDatabaseAgent”.

The low level database interface was implemented as a set of Fortran friendly procedures which could be called directly from the Fortran source code if required. This resulted in a number of odd data types in the procedure signatures (odd from a Delphi perspective).

The database interface was implemented in a Delphi unit “UFortranDatabaseInterface”. Most procedures required at least two parameters, a “Handle” and a Boolean result variable. The result variable indicated whether or not the procedure succeeded. The handle was an object address cast as an integer to allow it to be stored easily in the Fortran.

The object addresses referred to instances of the classes “TAccessDatabaseAgent” or “TDataSet”. The database agent was a custom class which encapsulated a Delphi ADO database connection “TADOConnection”. The reason that a custom class was used here was to allow the type of database connection to be changed later. The dataset was the standard Delphi abstract class “TDataSet”. The actual data set class used was TADOQuery. The abstract data set class was used to allow the actual type of data set to be changed at a later stage.

The data base agent class was implemented in a Delphi unit called “UAccessDatabaseAgent”. This file contains all the Delphi source code that implements that low level communication to the database. In the event that a new database type is required (not Microsoft Access) then this is the only file that needs to be re-developed.

At the time of the WR2005 project release, the database interface was not called directly from Fortran because it was possible to use high level Delphi procedures for everything.

9.20.3 Database Diagram

The dotted line boxes indicate groups of related tables. There are nine groups. Seven of the groups are self-explanatory as they relate to functional modules like reservoirs or mining. There are two master groups “Module Tables” and “Network Tables” where the “Network” table is the master of masters.

The “**Module Tables**” contain a unique description for every module in a given network. The module ID is in fact unique to the entire database.

The “**Network Tables**” contain the master content of a particular network. All data for a particular network link back to a single record in the Network table. A table called “NetworkModules” contains the list of all modules contained in a particular network. The data follows the following hierarchy:

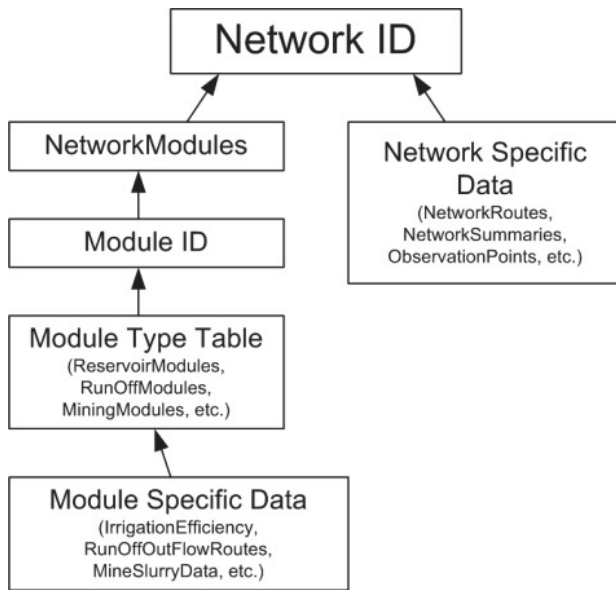


Figure 9-1: Database diagram

9.20.4 Graph Enhancement

In 2015, Mr Grant Nyland was commissioned to replace the existing graph utilities with a more powerful and user-friendly system which would allow for zooming, panning, changing to log scales, printing, reviewing data pertaining to the graph, etc. A graph manual is available on the system. This was done with a number of Delphi source code units as well as substantial changes to the following Lahey Fortran source code units:

- wrsm2000.f90
- wrsm2001.f90
- wrsm2014.f90
- wrsm2015.f90

New Delphi source code units for the graphs are as follows:

uChartPanel.pas
uChartReport.pas
uGraphForm.dfm
uGraphForm.pas
uGraphFormButtons.pas
uGraphType.pas
uPitmanChart.pas
uPitmanChartShapes.pas
uPitmanHistogramChart.pas
uPitmanLonLatChart.pas
uPitmanReservoirChart.pas
uPitmanScatterChart.pas
uPitmanSeasonalChart.pas
uPitmanTemporalChart.pas
uPitmanTimeSeriesChart.pas
uPitmanYieldChart.pas
uViewInv.pas

The Fortran file “WRSM2001.f90” which handles the graphs to be found in the Plot menu contains the subroutine “PlotSPF” which calls the subroutine Plot, i.e. “CALL Plot(graph)”. Likewise the Fortran file WRSM2015.f90 which handles the rainfall graphs also calls the subroutine Plot, i.e. “CALL Plot(graph)” which is in wrsm2014.f90 . The data for the graphs is prepared in the Fortran units, in particular wrsm2014.f90. The subroutine “GR_Draw” has a number of CASE statements handling the specific graph through (for example) the command “CALL GR_ExtractReservoirData(graph)”. Subroutines such as this one set up the arrays needed to plot the data. After the data is set up the command “CALL DoGraph(POINTER(graph))” calls the Delphi routine to do the actual plot. Numerous new Delphi source code units for the various graphs were created which are on the Delphi folder, for example uPitmanReservoirChart.pas. These source code units set the chart headings, chart properties, chart series and add chart data. If changes are required to titles for example, then the relevant Delphi source code unit is changed. The following schematic clarifies the logic and procedure for making changes.

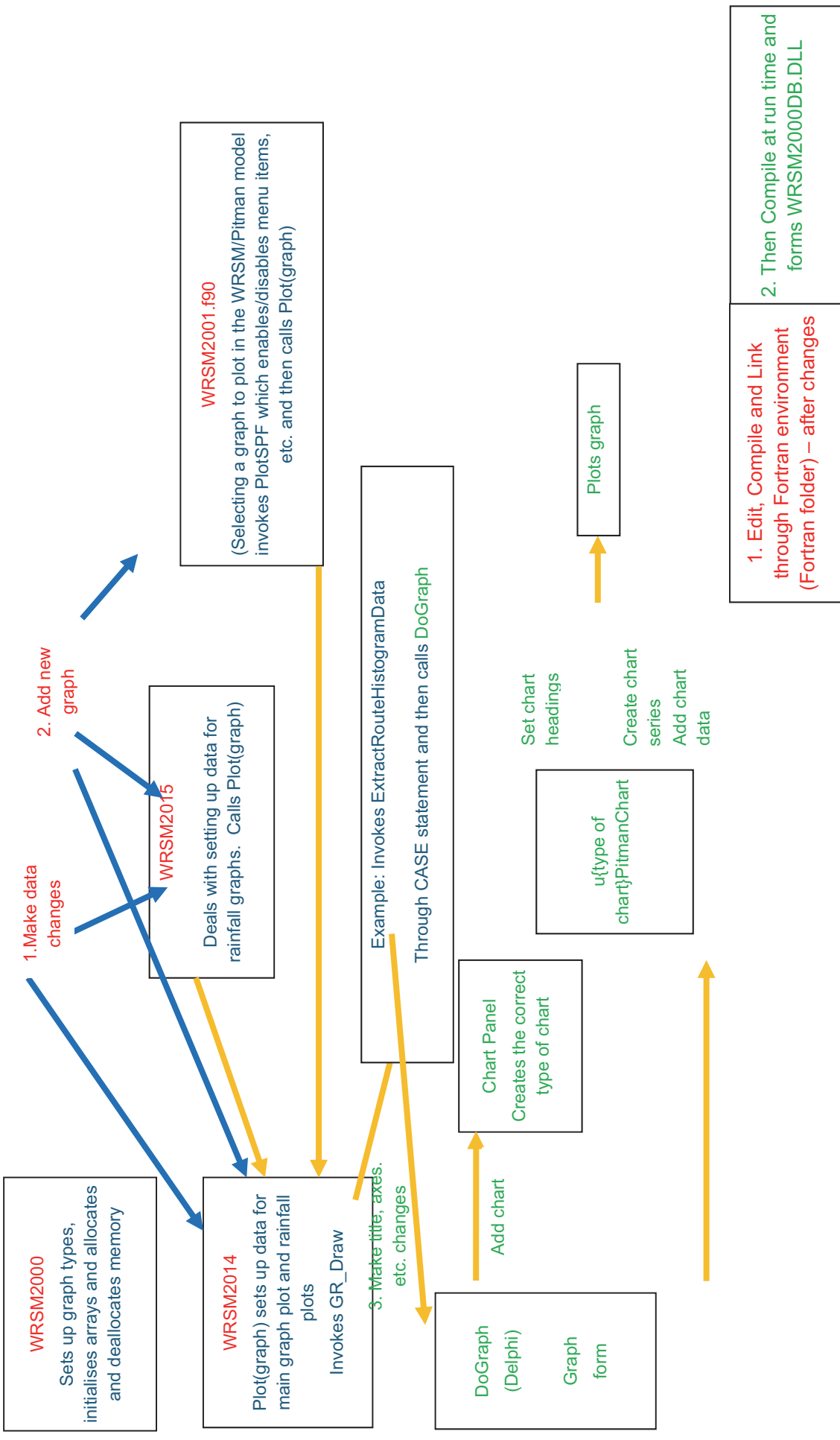


Figure 9-3: Graph logic and Procedures for changes to Fortran code and Delphi code

The equivalent Delphi file to the Fortran WRSM2000.WPJ is WRSM2000DB.dcu and is contained in the Delphi folder. Unlike Fortran, breakpoints can be set if the code is required to be debugged.

The Delphi code is compiled at run-time unlike the Fortran code. This Delphi code is contained in the WRSM2000DB.dll file.

Note that one of the Delphi subroutines handles the code security aspect of the program, namely: uLicenseAgent.pas .

Note also that there is a Logs folder that records error messages that a user could get.

9.20.5 Delphi Source Code Units

9.20.5.1 Low Level Database Units

Table 9.1: Delphi Source Code: Database

Delphi Source Code Unit	Purpose
UAccessDatabaseAgent.pas	Contains the class TAccessDatabaseAgent which implements the low level database interaction. This is where the actual table read writes occur.
UFortranDatabaseInterface.pas	Exported low level database functions. Allows for manipulation of queries and tables.

9.20.5.2 WRSM/Pitman Module Agent Units

Table 9.2: Delphi Source Code: WRSM/Pitman Module

Delphi Source Code Unit	Purpose
UChannelAgent.pas	Contains the Class TChannelAgent which implements channel related functionality.
UFlowFileAgent.pas	Contains the Class TFlowFileAgent which implements flow file related functionality.
UIrrigationAgent.pas	Contains the Class TIrrigationAgent which implements irrigation related functionality.
UMineAgent.pas	Contains the Class TMineAgent which implements mine related functionality.
UModuleAgent.pas	Contains the Class TModuleAgent which is an ancestor class common to WRSM/Pitman module classes. Functionality common to all modules is implemented here.
UNetworkAgent.pas	Contains the Class TNetworkAgent which implements network related functionality.
URainFileAgent.pas	Contains the Class TRainFileAgent which implements rain file related functionality.
URainGaugeAgent.pas	Contains the Class TRainGaugeAgent which implements rain gauge related functionality.
UReservoirAgent.pas	Contains the Class TReservoirAgent which implements

Delphi Source Code Unit	Purpose
	reservoir related functionality.
UResultFileAgent.pas	Contains the Class TResultFileAgent which implements result file related functionality.
URunOffAgent.pas	Contains the Class TRunOffAgent which implements run-off module related functionality.

9.20.5.3 Spatsim Related Units

Table 9.3: Delphi Source Code: Spatsim

Delphi Source Code Unit	Purpose
UMapObjectsAgent.pas	Contains the Class TMapObjectsAgent which implements functionality related to the MapObjects aspects of Spatsim. Spatsim uses MapObjects to implement its GIS functionality.
USpatsimAgent.pas	Contains the Class TSpatsimAgent which implements functionality related to Spatsim.

9.20.5.4 General Utility Related Units

Table 9.4: Delphi Source Code: General Utility

Delphi Source Code Unit	Purpose
UErrorHandlingOperations.pas	Contains general error management functions.
UFileReader.pas	A class that reads a file the same way that files are read in Fortran.
UListSelectionDialog.pas	Simple pop-up dialog that allows the user to select from a list.
UNumberLists.pas	Generic classes to manage a list of numbers.
UPopupDialogs.pas	Various general pop-up dialogs.
UProgressDialog.pas	A progress indication dialog.
UViewIni.pas	Stores program settings in the registry.

9.20.5.5 General Compile Notes

Difficulties were experienced with the Delphi version that Mr Allan Bailey had, so Delphi 7 was installed (an older version) which Mr Grant Nyland had developed with. Mr Grant Nyland also used TeeChartPro which was also installed (both installations require license codes which are obtained in license.txt). Mr Grant Nyland also had to tweak a few issues to get it to install. These are covered in the C:\Program Files (x86)\Steema code. Steema, Delphi v7 and TeeChartPro were all backed up on Mr Allan Bailey's external hard drive.

The WRSM2000 code if copied and linked only brings in the Delphi code (in WRSM2000DB.dll) at run time.

Changes to Fortran doesn't require any modifications to Delphi WRSM2000BD.dll. Changes to Delphi obviously do. A breakpoint can be set (line becomes red) in Delphi and one can step into and executecode line by line inspecting variables by placing the cursor on them.

9.21 Final compilation notes following the integration of SPATSIM and database features involving Delphi code, i.e. WRSM/Pitman Compilation Notes

This section contains notes on how to compile the WRSM/Pitman model.

9.21.1 Development Environments Required

The model required three development products – LF90, WiDE and Delphi.

9.21.1.1 Lahey Fortran 90 (LF90.exe)

The Lahey Fortran 90 compiler (version 4e) needs to be installed on the workstation and the bin directory must be in the workstation's PATH environment variable. Typically the compiler resides at "C:\LF9040" so "C:\LF9045\bin" must be in the PATH variable somewhere.

9.21.1.2 WiDE 2.2

The Winteracter compiler independent development environment "WiDE" version 2.2 is used to edit the source code. This application does not cope well with modern folder names so it must be installed at "C:\WINT". The path "C:\WINT\bin" must be added to the workstation's PATH variable.

9.21.1.3 Delphi 7

Borland's Delphi 7 is used to compile the DLL executable modules. This must be installed on the workstation. The path "C:\Program Files\Borland\Delphi7\Bin" should have been added to the PATH environment variable during the Delphi installation. This path is important if any command line compilations are to be attempted.

9.21.1.4 Changing the PATH for the above

Select Start

Select All Programs

Select Accessories

Select Windows Explorer

Right Click on My Computer

Select Properties

Select Advanced

Select Environment Variables

Select System Variable

Click on Path and Edit

Add C:\LF9045\bin;C:\WINT\bin and C:\Program Files\Borland\Delphi7\Bin to the path near the beginning

OK

OK

Now re-boot

Now copy LF9045 folder from back-up to the C drive root

Note that Delphi is not required for some modules

Note that WINT_Daily and WINT_New need C:\WINT_daily\bin and C:\WINT_new\bin .

This is changed by right-clicking on the WRSM2000.WPJ project file and opening with Notepad. Then change (for example) D:\WINT to the new C:\WINT_graph.

9.21.2 Compiled Output

The final product consists of three executables:

- **WRSM2000.EXE** – a Fortran program compiled with the Lahey Fortran 90 compiler (version 4e);
- **WREng.dll** – a Delphi DLL that contains the language strings. This is a Windows DLL that does not export any functions. Its sole purpose is to contain the resource strings used by the application. It was separated from the executable because the early resource compilers had a restriction on the number of resource strings. This DLL can be eliminated if the compiler is upgraded. Any version of Delphi from version 2 onwards can be used to compile this DLL and
- **WRSM2000DB.dll** – a Delphi DLL that implements the database functionality for the model. This is a Windows DLL that exports database related functions like “GetNetworkFromDatabase()”. The current implementation uses an ADO connection to a Microsoft Access database. Any version of Delphi from version 6 onwards can be used to compile this DLL.

All three files must reside in the same folder when the application is run.

9.21.3 Project Files

There are three project source code files (one for each of the executables) as follows :

- **WRSM2000.WPJ** – a WiDE project file that contains the project settings for WRSM2000.EXE. Open this project file with WiDE.
- **WREng.dpr** – a Delphi DLL project file to build the WREng.dll resource DLL. Open this file with Delphi and
- **WRSM2000DB.dpr** – a Delphi DLL project file to build the WRSM2000DB.dll database DLL. Open this file with Delphi.

9.21.4 WiDE Settings

When opening the WRSM2000.WPJ project file in WiDE the following should be displayed:

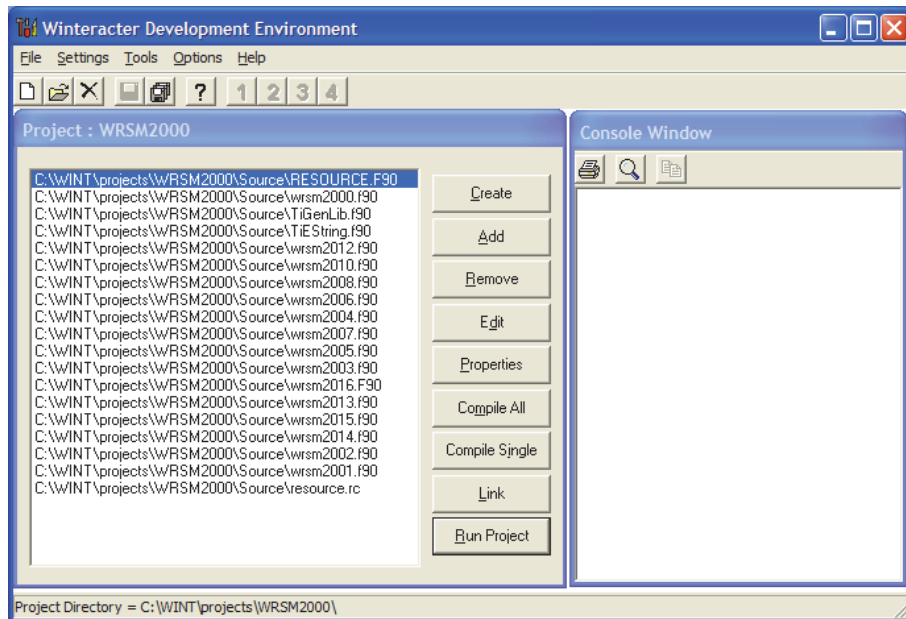


Figure 9-4: Winteracter Development Environment

9.21.4.1 Libraries

Link libraries are added through the menu [Settings] -> [Select Libraries].

- add all the libraries in the path C:\WINT\lib.lah to the library list in the “Select Link Libraries” dialog. These libraries would apply to all projects that use LF90 and
- also add one library from the project bin subfolder called “tisd.lib”. This file must be in the folder for the link to succeed. This library is only used by this project.

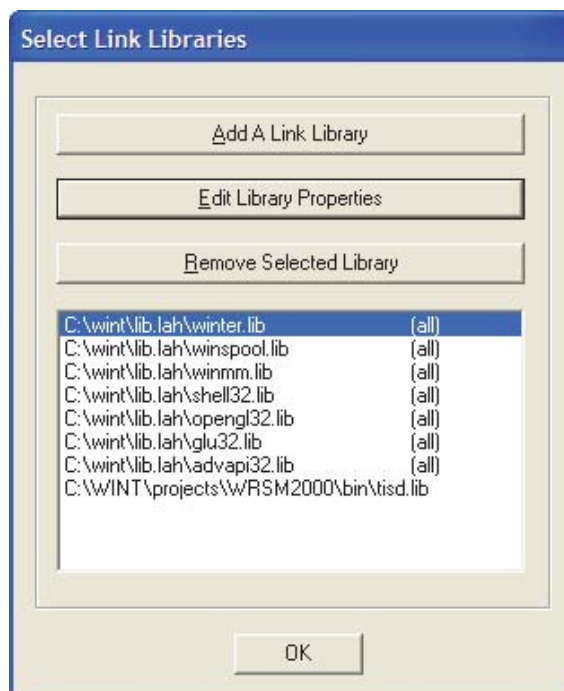


Figure 9-5: Libraries

9.21.4.2 Compiler Options

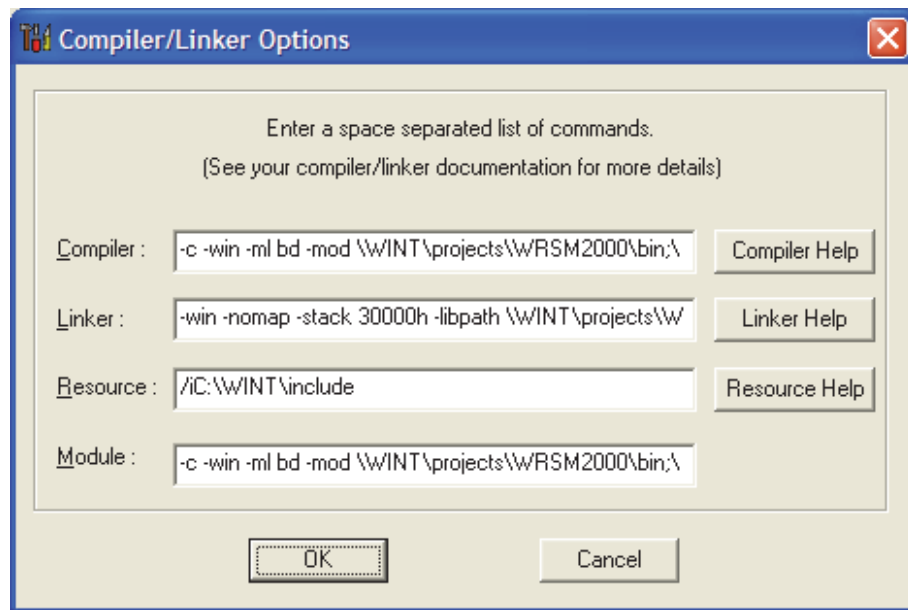


Figure 9-6: Compiler Options

Compiler Directives

These flags are set through the menu [Settings] -> [Compiler/Linker Options]. These flags must be set for both modules and normal Fortran files. Both edit boxes (Compiler & Module) must therefore contain the same string indicated below.

Compiler flags: **-c -win -ml bd -mod \\WINT\projects\WRSM2000\bin;\WINT\lib.lah**

-c

This directive instructs the compiler to compile only – no linking. This should be the first flag present in the compiler flags in all applications in WiDE because WiDE has a separate “Link” button on the screen.

-win

This directive instructs the compiler that the application is a Windows application. WRSM/Pitman is for Windows so this flag must be present. This directive is also required because DLL’s are used and DLL’s are purely Windows constructions.

-ml bd

This directive instructs the compiler to expect DLL’s compiled by Borland’s Delphi. It enforces the “stdcall” calling convention. Functions exported from Delphi must be declared as stdcall. When calling a Fortran function in Delphi then Delphi must also expect the function to be stdcall.

-mod \\WINT\projects\WRSM2000\bin;\WINT\lib.lah

This directive instructs the compiler where to look for additional library modules. The first path is very important because this is where the modules from the application will be placed. If this is left out the applications modules will be placed in the library folder which could corrupt the library for other applications. The second path indicates that the Lahey Fortran 90 libraries are to be used. Be sure to create a bin subfolder in the project folder.

Linker Directives

These flags are set through the menu [Settings] -> [Compiler/Linker Options]. They are contained in the edit box called "Linker". For this version the linking is also done by the compiler. Both compilation and linking commands are sent to LF90.EXE in two separate steps. The compiler passes the linking parameters to the linker.

-win -nomap -fullwarn -stack 30000h -libpath \WINT\projects\WRSM2000\Bin -implib WRSM2000DB.dll -import DoDebug DoPopup DoShell GetNW SaveNW DeINW GetRG SaveRG DeIRG SaveToSptsm VerifyLicense

-win

This directive instructs the linker that the application is a Windows application. WRSM/Pitman is for Windows so this flag must be present. This directive is also required because DLL's are used and DLL's are purely Windows constructions.

-nomap

This directive instructs the linker not to generate a map file.

-stack 30000h

This directive instructs the linker to create a 30 Megabyte stack. It is not clear why the default was changed and a larger stack chosen.

-libpath \WINT\projects\WRSM2000\bin

This directive instructs the linker to look for the lib files in the bin subfolder.

-implib WRSM2000DB.dll -import GetNW

This directive instructs the linker to import the functions from the DLL. All function names should be listed. GETNW is to do with getting the network from the database.

Similarly **SaveNW** and **DELNW**

GetRG, SaveRG, DeIRG Related to saving rainfall data in the database – not used at present

SaveToSpatsm

Spatsim option

VerifyLicense

Allows for control of the use of WRSM/Pitman by using a code generated from the serial number of the user's PC/laptop computer.

Resource Directives

These flags are set through the menu [Settings] -> [Compiler/Linker Options]. They are contained in the edit box called "Resource".

/iC:\WINT\include

This directive instructs the linker to look in that folder for additional header files.

Module

These flags are set through the menu [Settings] -> [Compiler/Linker Options]. They are contained in the edit box called "Module".

```
-c -win -ml bd -mod \WINT\projects\WRSM2000\bin;\WIN\lib.lah
```

Executable Filename

This setting is set through the menu [Settings] -> [Select Executable Name].

WRSM2000.EXE

This setting instructs the linker to name the linked executable accordingly. The path is optional but WiDE will put the path in itself.

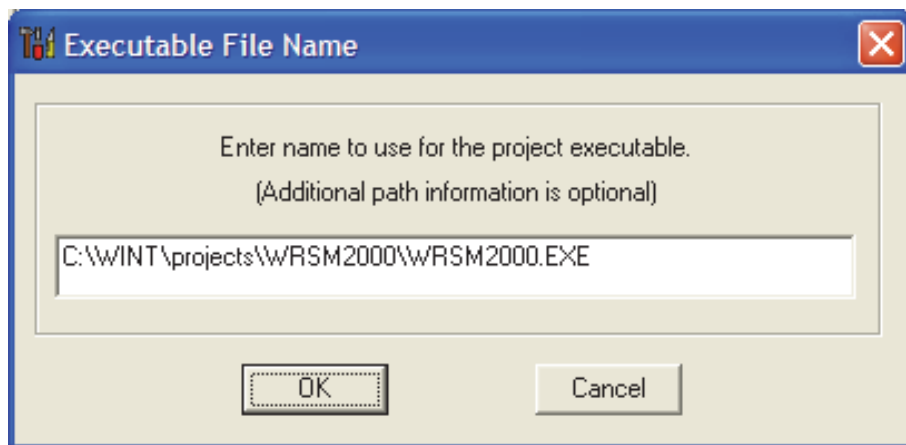


Figure 9-7: Executable

9.21.4.3 Circular References

The compiler has a problem with circular references. This application has circular references between the stats_module and other modules. The only way to get the system to compile is to do the following:

WiDE creates a file called "**modtable.txt**" in the compile output bin subfolder. The name of every module created and the related source code file name is placed in this file automatically as the files are compiled. This file is not supposed to be edited. Edit this file and place the following line at the top of the file just below the last comment line:

```
stats_module stats_m0  
C:\WINT\PROJECTS\WRSM2000\SOURCE\WRSM2013.F90
```

Locate a previously compiled copy of the file "stats_m0.mod" and place it in the compiler output bin subfolder. The compiler uses the old copy to compile the dependent files before recompiling that file. The user should never have to change the interface of any function in stats_module!

9.21.4.4 Fortran File Properties

These settings are set by clicking on the source code file and then clicking on the [Properties] button.

1. All files that contain normal Fortran but do not contain the MAIN() function must have properties as below. This means that the compiler will compile the file and make a mod file. These files are not linked because only the file that contains the main routine will be linked.

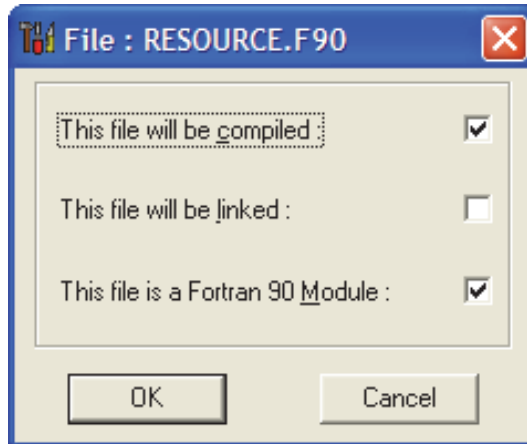


Figure 9-8: "Resource.F90" Compilation

2. The file that contains the MAIN() function "wrsm2001.f90" must have properties as below. This means that the compiler will compile the file and then link all the other Fortran into one application.

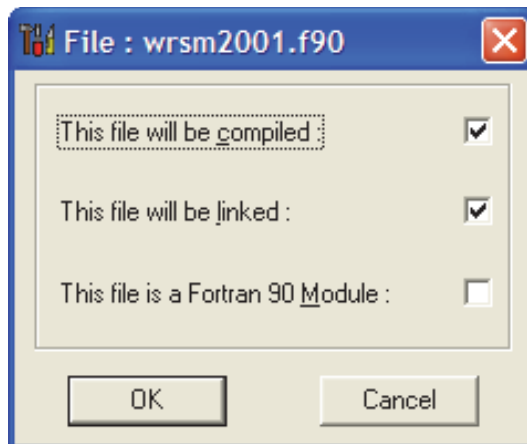


Figure 9-9: Compilation

3. Resource files "*.rc" must have properties as below. This means that the compiler will pass the file to a resource compiler (this is not a Fortran file). The resulting "*.res" file that is produced will be linked into the final application.

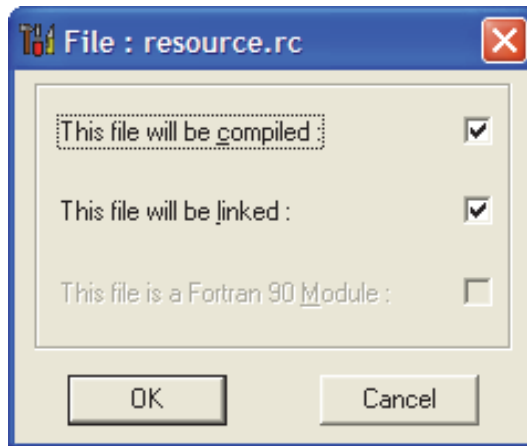


Figure 9-10: "Resource.rc" Compilation

9.21.4.5 Delphi Settings

No special settings are required for the Delphi DLL's. All functions exported from or imported into the Delphi DLL's must be declared with the "stdcall" calling convention.

9.22 License

In May 2005 a software licence system was implemented due to the fact that WR2005 DVD includes WRSM/Pitman and it is sold as a software product. The software license is a key linking the user's version of WRSM/Pitman to his or her PC/laptop serial number. The procedure is that the user will provide Allan Bailey with their serial number. Allan Bailey will generate a code using the program WRSM2000LicenseGenerator once the user has paid for the model. The user will be prompted for this number on entering the program but can still enter data. WRSM/Pitman will, however, not run until the correct code is entered.

10 MONTHLY AND DAILY TIME STEP DIFFERENCES

Version 2.9 of WRSM/Pitman should be used for the monthly time step and version 2.6 for the daily time step. The main difference in the cde is that for the daily time step, a new function was added – RU_Calculate_daily . This function as taken from the DOS model called DAILY_P . There is a tickbox on the Network File | Load Network screen that the user should tick for a daily time step. This will then invoke RU_Calculate_daily instead of RU_Calculate which is for the monthly time step.

There is a completely new screen in the runoff module, namely Calibrate Daily which was set up in the resource.rc module as IDD_RU_EDIT_CALIBRATE_DAILY . A number of other screens had to be changed in the resource.rc module to deal with daily rainfall plotting graphs.

There is another function in wrsm2004.f90 which had to be created, namely : RU_GetValueDaily which returns the runoff module number.

In wrsm2002.f90 a number of system functions were required such as:

- SY_GetDailyTStep;
- SY_ShowDailyLandUse;
- SY_CalculateDailyLandUse;
- SY_CalculateSimDailyFlow;
- SY_AddNatDailyFileName;
- SY_DeINatDailyFileName
- SY_Daily_rain_format_conv
- SY_Daily_flow_format_conv

In wrsm2014.f90 two new graphs were added as follows:

- GR_DailyHydrograph(graph) for a daily time step hydrograph for naturalised runoff modules
- GR_DailyHydrographLandUse(graph) for a daily time step hydrograph for land use

In wrsm2001.f90 a subroutine was added to invoke SY_ShowDailyLandUse, namely: DailyLandUse .

In wrsm2000.f90 a number of variables were added to graph_Type to deal with daily graphs. Only 2 graphs are possible with the daily time step so all monthly graph options are greyed out in the daily time step version.

In wrsm2010.f90 the following functions were added to deal with rainfall (note that some functions had Daily added and some just “_d” added to signify a daily equivalent to a monthly function of similar name):

- RG_ReadDaily
- RG_GetValueDaily
- RG_SetValueDaily
- RG_GetValueDaily
- RG_SetValueDaily
- RG_ReadDaily
- RG_GetFileN_d
- RG_DeleteAll_d
- RG_Delete_d
- RG_ShowAll_d
- RG_DeleteAll_d
- RG_Delete_d
- RG_ShowAll_d
- RG_GetName_d and

- RG_GetName_d .

In wrsm2012.f90 the following functions were added to deal mainly with the start and end of the daily record period which has to be contained within the monthly time period:

- SG_GetYStart_d
- SG_GetYEnd_d
- SG_SetYStart_d
- SG_SetYEnd_D
- SG_GetYStart_d
- SG_GetYEnd_d
- SG_SetYStart_d
- SG_SetYEnd_d
- SG_GetDaily_d and
- SG_SetDaily_d .

11 SPECIAL NOTES

11.1 Calibration parameters

There is an array which keeps track of the calibration parameters for each of the three groundwater methods, namely: Pitman, Hughes and Sami. These variables have to be kept separate because they can have different meanings. The array is called (using ZMin as an example) :

```
Runoff_PA(iselect) % Runoff_PTR % A_ZMIN(i)
```

With

- i = 1 for Pitman,
- i = 2 for Hughes and
- i = 3 for Sami

When the user presses the Apply key in Edit, the calibration parameters (along with everything else) are saved to memory. The following array keeps track of the calibration parameters in memory.

```
Runoff_PA(iselect) % Runoff_PTR % ZMIN(i)
```

If the user wants to save a network to retain these calibration parameters, the function **Save** is called which in turn calls the function **Write**. Within **Write**, the function **LoadParameters** is called with the relevant groundwater model method to write out the relevant parameters to file. At the end of the **Write** function, **LoadParameters** is invoked to re-set memory to the current groundwater method. This is required in case the user applies data that he/she has entered, saves the network but then goes back into Edit.

11.2 SAMI Analysis

In a Sami analysis, the Pitman analysis is called first by means of the RU_GW function i.e. with both Pitman and Hughes groundwater models, the RU_GW function is called. Prior to this, if the Sami method is being used, GW and GL (Pitman groundwater parameters) are set to zero (globally) so that there is no effect of GW and GL on the Sami analysis.

11.3 Screen display – colours

Colours are made up of percentages of three primary colours, red, blue and green. Use can be made of the software program PAINT to select the intensity of the three colours making up the final colour. 256 is the maximum intensity for a colour and 0 is the lowest. Black for example is 256 for green, red and blue. This is set up in WRSM2004.F90 . Colours have been selected for the three classes of parameter, namely : white, blue and pink. White is for category 3 – those parameters which should be changed, blue for category 2 – those that could be changed if you have the necessary information and pink for category 1 – those that should not be changed. These categories are given in a table the User Manual – section 9.

The statement :

```
CALL WDialogColour(IDF_HGSL, forecol, rbackcol)
```

Sets the colour to blue for example.

For those parameters that never change , for example, FF, use can be made of the Winteracter environment by going into resource.rc with the Resource editor, choosing IDD_RU_EDIT_GWSAMI for example and right clicking on the parameter/field, selecting Colour, Background, Specified, Select and

changing there.

The code will overrule anything that is set up in the resource.rc file

11.4 Making fields inactive

This was required for example with the calibration parameters GW and GL which are not required for the Sami model.

The statement :

```
CALL WDialogFieldState(IDF_GW, 0)
```

Sets the GW parameter to off i.e. the field will be greyed out and the user will not be able to change it. A 1 sets it to on, making it active i.e. a certain colour and the user able to change the value. This is done in the code where calibration parameters can be on or off depending on the groundwater method.

The statement :

```
CALL WDialogClearField(IDF_GW)
```

Clears whatever value may be in the field (on the screen)

For those parameters that never change , for example, FF, use can be made of the Winteracter environment by going into resource.rc with the Resource editor, choosing IDD_RU_EDIT_CALIBRATE and right clicking on the parameter/field, changing in Style and checking the Read Only tab .

The code will overrule anything that is set up in the resource.rc file

11.5 Compiler/linker options

In the development environment, the following options should be set in WINT under Settings and Compiler/Linker options

Compiler	-c -win -ml msvc -mod .;\wint\lib.lah
Linker	-win -nomap -fullwarn -stack 30000h -ml msvc -libp \wint\lib.lah
Resource	/i\wint\include
Module	-c

Also if one goes into My Computer (right click), Properties , Advanced, Environment variables, System variables

The path should be

```
C:\WINT\BIN;%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;C:\Program Files\ATI Technologies\ATI Control Panel;C:\LF9045\BIN;
```

The wide.ini file is also relevant with regard to libraries

11.6 Adding rows or columns to screen tables

This was done in the case of Irrigation in the CROPS2 screen where the crop description was to be added. Firstly the resource editor must be invoked to add the extra column. Choose the appropriate dialog and

right click on the table. In the columns section, select New and name the extra column and put it in the correct place.

Now the code has to be added. Add a variable (described in wrsm2007 as akb17july2006), initialise it, add the variable to the Read statement (this should be at the end of the line to minimise the difficulty in adding variables.- datafiles need to have a manual insertion of 'Sugar Cane' or whatever), write the variable to the output file, clear the screen grid, write to the screen, read the data from the screen grid and add to the debug file if necessary.

11.7 Adding a new graph – Groundwater

This was done for example with the groundwater plot. Firstly go into the "resource.rc" datafile and choose the IDD_PLOT dialogue. Extend the window and select a new radio button and text box. Then make space and select a new combo box so that the required runoff module can be selected once the groundwater – surface water plot is chosen. For this plot it required a runoff module as opposed to a route which is required for a number of other plots. The "Net catchment runoff": and "groundwater outflow" were plotted i.e. groundwater flow and total surface + groundwater flow (as can be seen if Save| Runoff Module| etc. is chosen from the File pull-down menu for time series.). This new radio button is given a meaningful name (IDF_GSP_RADIO in this case). Very important note : When the new radio button is added, for some reason (bug) it does not get included in the group and clicking on it will not deactivate the other radio button that is activated and all the relevant code pertaining to that the new radio button will not be invoked. What needs to be done is to go into resource.rc in NOTEPAD, find the second occurrence of IDD_PLOT, find the latest radio button code at the bottom of the section, cut it and paste it higher up directly underneath the previous radio button code. Delete the line at the bottom where you cut out the code. Now the radio button will be included in the group and you can click on it correctly and invoke the necessary code.

Now a number of sections of code in WRSM2001.F90 need to be included/changed. This can be checked on by searching for the comment with "akb17aug2006. Also WRSM2014 required an additional function ("GR_Groundwater"). Another function was also required to get the values for the groundwater and net catchment runoff from WRSM2014.F90. This function was called "RU_GetValue" and was included in WRSM2004.F90. "RU_GetValue and was used in the "GR_Groundwater" function to get the values to be plotted in the required variables.

Significant changes were required in Subroutine "PLOTSPF" in WRSM2001.F90 to enable the runoff module selection and disable the others and also to set up "rumods" which enables the user to select which runoff module they want to plot.

Titles, legends, etc. were altered in the "SCPLOT" function in WRSM2014.

If the groundwater values were small negatives, these were set to 0 (for plotting only).

11.8 Adding a new graph and adding to a data structure type – Catchment Based Rainfall Massplot

The catchment based massplot graph also required that a new variable be set up in a rainfall data structure. The reason was as follows :

WRSM2015.f90 checks for the button "Catchment Based Rainfall Massplot" to be activated. Then the PLOTTRFD and PLOT routines are called (from WRSM2001.f90) which in turn called the GR_Draw function in WRSM2014.f90. A new function GR_ExtractRainfallMassplotData was established in wrsm2014 which is called by GR_Draw and which then finally calls the Delphi function DoGRaph to actually plot the graph. The problem then is that a function which had been set up in WRSM2015 had to pass an array to

WRSM2014. The solution was to add an array to the rainfall data structure R2_Data in a similar way to "frequency" for the Temporal distribution plot. This entailed adding CTOT to the 2_Data structure in wrsm2000.f90, then using the existing R2_Calculate function in wrsm2015.f90 to include code to set up the cumulative total percentage for the catchment based rainfall datafile that had just been determined (in R2_Calculate), then writing a new function R2_GetCTOT (very similar to R2_GetFrequency) in wrsm2015.f90 and finally setting up the YValues year by year using the R2_GetCTOT function GR_CatchmentRainfallMassplot(graph) in wrsm2014.

The button "Catchment Based Rainfall Massplot" also had to be set off until the Calculate button had been pressed.

11.9 Changing the SSI logo

Simply replace the datafile "ssi.bmp" in the WINT/PROJECTS/WRSM2000 directory and re-compile

11.10 Adding radio buttons

Radio buttons have to be grouped so that if the user clicks on one of them then the other(s) become unclicked. In the resource editor, copy and paste a radio button on to the appropriate form, change description by right-clicking, etc. **IMPORTANT NOTE** : Then go into "resource.rc" but choose the text editor and move the last line of the appropriate dialog up one row so that the command line with the group statement comes last. In the example below the WQT-SAPWAT method was added by copy and pasting a radio button. The second last line with the WQT-SAPWAT description had to be moved above the line "CONTROL "Model Type ...)" otherwise one could have two radio buttons ticked at the same time i.e. they would not be grouped. Look for the IDD_PLOT_DIALOG for other examples like the following

```
CONTROL "Flow Storage Yield plot",IDF_SMP_RADIO3 .. etc.
```

This statement had to be moved from the last row in this block up to where it is now.

```

IDD_RR_EDIT_GENERAL_DIALOG 0, 0, 240, 220
STYLE WS_CHILD | DS_CONTROL | DS_3DLOOK
FONT 8, "MS Sans Serif"
BEGIN
CONTROL "Module Name",IDF_LABEL1,"STATIC",WS_CHILD | WS_VISIBLE | WS_GROUP | SS_LEFT, 2, 12, 62, 8
CONTROL "",IDF_RR_NAME,"EDIT",WS_CHILD | WS_VISIBLE | WS_BORDER | WS_GROUP | WS_TABSTOP | ES_LEFT | ES_AUTOHSCROLL, 71, 8, 130, 14
CONTROL "",IDF_MOD_FILENAME,"EDIT",WS_CHILD | WS_VISIBLE | WS_DISABLED | WS_BORDER | WS_GROUP | WS_TABSTOP | ES_LEFT | ES_AUTOHSCROLL, 71, 28, 130, 14
CONTROL "0",IDF_ABSTRACTION_ROUTE,"INTEGEREDIT",WS_CHILD | WS_VISIBLE | WS_BORDER | WS_GROUP | ES_LEFT | ES_RIGHT | ES_MULTILINE | ES_READONLY, 102, 48, 40,
14
CONTROL "0",IDF_RETURN_FLOW_ROUTE,"INTEGEREDIT",WS_CHILD | WS_VISIBLE | WS_BORDER | WS_GROUP | ES_LEFT | ES_RIGHT | ES_MULTILINE | ES_READONLY, 102, 68, 40,
14
CONTROL "Module file name",IDF_LABEL2,"STATIC",WS_CHILD | WS_VISIBLE | WS_GROUP | SS_LEFT, 2, 32, 62, 8
CONTROL "Return Flow Route",IDF_LABEL4,"STATIC",WS_CHILD | WS_VISIBLE | SS_LEFT, 2, 72, 100, 8
CONTROL "Abstraction Route",IDF_LABEL5,"STATIC",WS_CHILD | WS_VISIBLE | WS_GROUP | SS_LEFT, 2, 52, 100, 8
CONTROL "Original WRSM 2000",IDF_RADIO1,"BUTTON",WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP | BS_AUTORADIOBUTTON | BS_LEFTTEXT | BS_TEXT, 78, 124, 100, 14
CONTROL "WQT model",IDF_RADIO2,"BUTTON",WS_CHILD | WS_VISIBLE | WS_TABSTOP | BS_AUTORADIOBUTTON | BS_LEFTTEXT | BS_TEXT, 80, 140, 98, 14
CONTROL "WQT-SAPWAT model",IDF_RADIO3,"BUTTON",WS_CHILD | WS_VISIBLE | WS_TABSTOP | BS_AUTORADIOBUTTON | BS_LEFTTEXT | BS_TEXT, 80, 160, 98, 12
CONTROL "Model Type",IDF_GROUP2,"BUTTON",WS_CHILD | WS_VISIBLE | WS_GROUP | BS_GROUPBOX | BS_TEXT, 74, 108, 114, 68
END

```

Take note of the name for the radio button (in resource.rc). For WQT-SAPWAT it is IDF_RADIO3 . This name must then be referenced in the code and a variable set up to represent it (Irrig_PA(nc) % Irrig_PTR % FILETYPE = 3) . See example below.

```
CALL WDialogGetRadioButton(IDF_RADIO1,IPOSITION)
SELECT CASE (IPOSITION)
CASE(1)
  if( Irrig_PA(nc) % Irrig_PTR % MODEL .NE. 1 )then
    Irrig_PA(nc) % Irrig_PTR % CHANGED = .TRUE.
    ! If the user changed the model, write the file back as a
    ! type 3 file that contains both models.
    Irrig_PA(nc) % Irrig_PTR % FILETYPE = 3
    Irrig_PA(nc) % Irrig_PTR % MODEL = 1
    fresult = RR_SetActives(nc)
  end if
CASE(2)
  if( Irrig_PA(nc) % Irrig_PTR % MODEL .NE. 2 )then
    Irrig_PA(nc) % Irrig_PTR % CHANGED = .TRUE.
    Irrig_PA(nc) % Irrig_PTR % MODEL = 2
    ! If the user changed the model, write the file back as a
    ! type 3 file that contains both models.
    Irrig_PA(nc) % Irrig_PTR % FILETYPE = 3
    fresult = RR_SetActives(nc)
  end if
! akb3nov2006 Add WQT-SAPWAT method
CASE(3)
  if( Irrig_PA(nc) % Irrig_PTR % MODEL .NE. 3 )then
    Irrig_PA(nc) % Irrig_PTR % CHANGED = .TRUE.
    Irrig_PA(nc) % Irrig_PTR % MODEL = 3
    ! If the user changed the model, write the file back as a
    ! type 3 file that contains both models.
    Irrig_PA(nc) % Irrig_PTR % FILETYPE = 3
    fresult = RR_SetActives(nc)
  end if
END SELECT
```

Further note on Radio Buttons

When adding a groundwater abstraction time series option, i.e. 0= no groundwater abstractions, 1= annual and 2=Time series, despite moving the rows on the next page up in the order, it did not work. The text had to be manipulated as follows on the next page . There are 3 radio buttons and a group box. Note the following :

1. The WS_GROUP must only be on the first radio button
2. The three lines must be in the correct radio button grouping
3. The numbers 12, 136, 50 and 14 must be the same for all except for the second number
4. The group line must come last
5. BS_AUTORADIOBUTTON must be in all 3

CONTROL "None",IDF_Radio_GW_None,"BUTTON",WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP | BS_AUTORADIOBUTTON | BS_TEXT, 12, 136, 50, 14
CONTROL "Annual",IDF_RADIO_GW_AN,"BUTTON",WS_CHILD | WS_VISIBLE | WS_TABSTOP | BS_AUTORADIOBUTTON | BS_TEXT, 12, 148, 50, 14
CONTROL "Time Series",IDF_RADIO_GW_TS,"BUTTON",WS_CHILD | WS_VISIBLE | WS_TABSTOP | BS_AUTORADIOBUTTON | BS_TEXT, 12, 160, 50, 14
CONTROL "GW Abs",IDF_GROUP_GW,"BUTTON",WS_CHILD | WS_VISIBLE | WS_GROUP | BS_GROUPBOX | BS_TEXT, 6, 128, 68, 48

11.11 Adding a tick-box

Copy and paste a tickbox or establish one from scratch.

In the routine EDIT_GS, one needs a WDialogGetCheckBox and WDialogPutCheckBox as follows :

```
CALL WDialogGetCheckBox(IDF_DROUGHT_APPLICABLE, Irrig_PA(nc) % Irrig_PTR %  
DROUGHT_APPLICABLE)
```

```
CALL WDialogPutCheckBox(IDF_DROUGHT_APPLICABLE, Irrig_PA(nc) % Irrig_PTR %  
DROUGHT_APPLICABLE)
```

These statements link the screen checkbox (IDF_DROUGHT_APPLICABLE) to the parameter used in the code (Irrig_PA(nc) % Irrig_PTR % DROUGHT_APPLICABLE)

11.12 Transferring certain variables using a GET routine

This was required for the WQT-SAPWAT method where the total rain per year and average rain were not available. They had to be calculated in WRSM2010.f90 and transferred to WRSM2007.f90 . A function RG_GetAnp had to be added (also RG_GetPerc) toGrant

```
! akb10nov2006 Added RG_GetPerc and RG_GetAnp
```

```
! =====
```

```
Function RG_GetAnp(aExternalGaugeNumber, aYearIndex)
```

```
! This function returns anp (average annual rainfall in mm)
```

```
! This is a Get function to make anp read-only
```

```
! =====
```

```
implicit none
```

```
INTEGER, INTENT(IN) :: aExternalGaugeNumber, aYearIndex
```

```
REAL :: RG_GetAnp
```

```
INTEGER :: I_internalGaugeIndex
```

```
I_internalGaugeIndex = RG_GetIndex(aExternalGaugeNumber)
```

```
RG_GetAnp = Raingauge_PA(I_internalGaugeIndex) % Raingauge_PTR % anp(aYearIndex)
```

```
return
```

```
end function RG_GetAnp
```

11.13 SPATSIM database settings

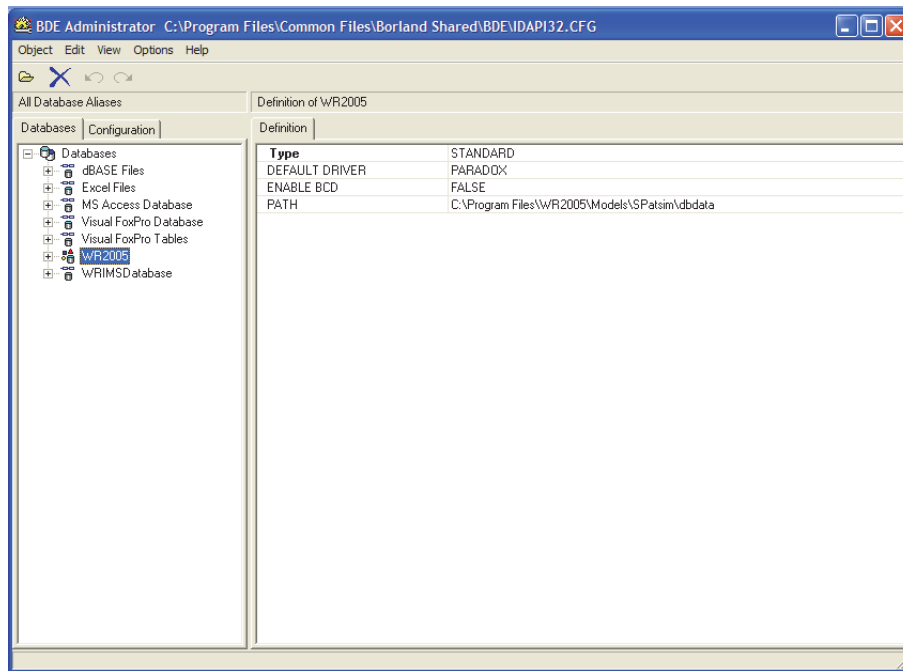


Figure 11-1: BDE File Administrator

The WR2005 database has to be set up as shown in order to load the SA map and allow features and attributes to be set up, etc. This can be accessed from any Borland BDE Administrator menu item, e.g. Borland 3 or Codegear/RAD Studio/5.0/bin/dbexplor.exe . Then right-click on Database, select New and enter WR2005 as the database driver name. Then set up properties as follows :

Type	:	STANDARD
Default Driver	:	PARADOX
Enable BCD	:	False
Path	:	c:\Program Files\WR2005\Models\Spatsim\dbdata

Save these settings, this will attach data dictionaries, etc. as follows :

```
array_data.DB
data_dict1.DB
data_dict2.DB
data_dict3.DB
data_dict4.DB
integer_data.DB
memo_data.DB
proc_run.DB
real_data-DB
text_data-DB
ts_data.DB
```

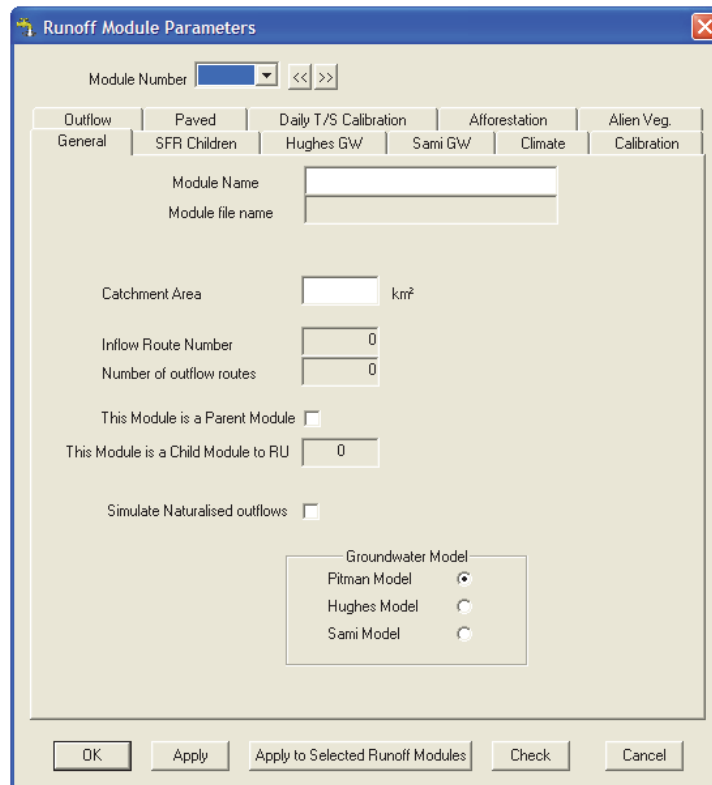
Then re-start the laptop, go into Spatsim and select Features/Add/Existing and choose Quaternary Catchments and Finished. Then select Features/ID or Desc Field/Edit Fileds/NATRUNOFF. Finally selecting the small A will insert the MARs for all quaternary catchments.

Now go into the dashboard

11.14 Establishing new dialogs

This was required for “Grouping of runoff modules for multiple calibration changes”. One of the most difficult aspects of this was setting up relevant structures so that when buttons were pressed, the correct actions would follow. Grant Nyland assisted with this.

Firstly a new button was added to the Edit Runoff window as shown below (Apply to Selected Runoff Modules).



The image shows a software dialog box titled "Runoff Module Parameters". At the top, there is a "Module Number" dropdown menu and navigation buttons "<<" and ">>". Below this is a tabbed interface with tabs for "Outflow", "Paved", "Daily T/S Calibration", "Afforestation", and "Alien Veg.". The "General" tab is selected, showing sub-tabs for "General", "SFR Children", "Hughes GW", "Sami GW", "Climate", and "Calibration". The "General" sub-tab contains the following fields and controls:

- Module Name: text input field
- Module file name: text input field
- Catchment Area: text input field followed by "km²"
- Inflow Route Number: text input field with "0" inside
- Number of outflow routes: text input field with "0" inside
- This Module is a Parent Module: checkbox (unchecked)
- This Module is a Child Module to RU: text input field with "0" inside
- Simulate Naturalised outflows: checkbox (unchecked)
- Groundwater Model section with three radio buttons:
 - Pitman Model (selected)
 - Hughes Model
 - Sami Model

At the bottom of the dialog are five buttons: "OK", "Apply", "Apply to Selected Runoff Modules", "Check", and "Cancel".

Figure 11-2: Runoff Module Parameters

Pressing this button activates the new dialog shown below.

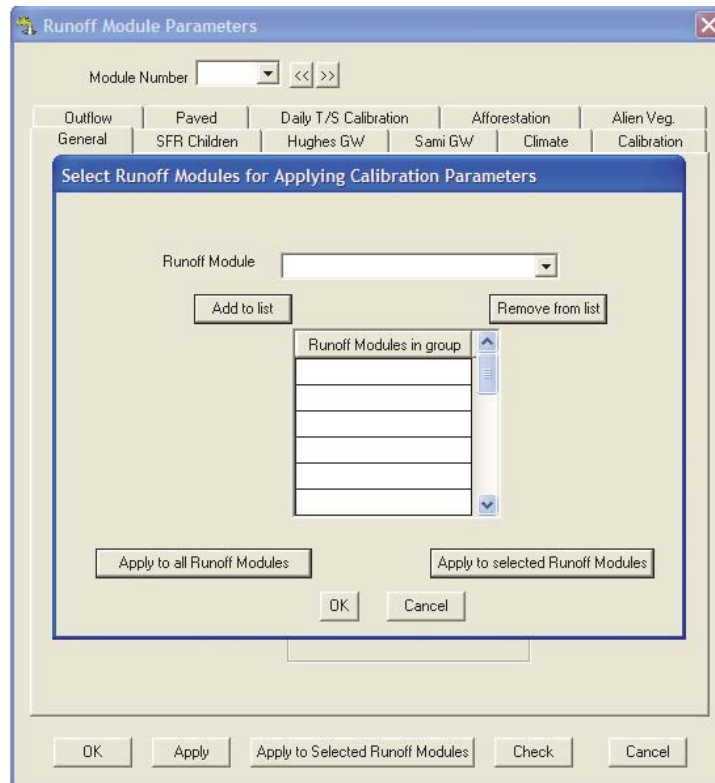


Figure 11-3: Runoff Module: Multiple Calibration

These buttons, input windows and grids are assigned a number when they are created. For example the grid with the heading “Runoff Modules in group” above has the number 1461. You can check on this by editing the dialog under the Text Editor and searching for this number .. You will see the command :

```
#define IDF_RUNOFF_CAL_GRID          1461
```

Meaning that the grid with the name IDF_RUNOFF_CAL_GRID has the number 1461. If you right click on the grid under Edit/Resource.rc/DialogEd/Choose Dialog (then selectIDD_Group_Runoff_Modules)/then right click on the grid/General AltEnter/ID

The procedure that was adopted was to set up a construct just to show the dialog above first in a loop along with other buttons to Save a single module, etc. and then in another loop to perform actions such as selecting a runoff module from the menu option and placing it in the grid.

! Get the next lot of messages.

```
CALL WMessage(ITYPE,MESSAGE)
```

! Process events from the Run Off Edit Dialog.

```
if(MESSAGE%WIN.EQ.IDD_RU_EDIT)then
```

! Branch on the type of event.

```
SELECT CASE (ITYPE)
```

! Process Button Clicks.

```
CASE (PushButton)
```

! Branch on the buttons on the dialog.

```
SELECT CASE (MESSAGE%VALUE1)
```

```

! Save Button.
CASE (IDF_SAVE)

! The user pressed the APPLY button to Save the Runoff Module
! Currently in the dialog. Save the stuff, but don't close the
! dialog
fresult = RU_EDIT_GS(nc,2)
if( fresult > 0 )then
! Set the active cursor button to 'Check'
CALL WDialogSetField(IDF_CHECK)
end if

! Save the parameters in the proper array.
if( RU_SaveParameters(nc) > 0 )then
end if

! Set the values to those in the memory
fresult = RU_EDIT_GS(nc,1)

! "Apply To All or Some" Button
CASE (IDF_SAVE_ALL_R)
CALL EH_GroupRunOffModules_Show(ITYPE,MESSAGE)

```

Event handlers such as EH_GroupRunOffModules_Show were set up as subroutines in wrsm2004.

Then the other loop to perform actions is as shown below.

! akb30june2011 Added by Grant for runoff groups to End of Dialog handle

```

! Process events from the Group Run Off Modules Dialog.
if(MESSAGE%WIN.EQ.IDD_GROUP_RUNOFF_MODULES)then

! Branch on the type of event.
SELECT CASE (ITYPE)

! Process Button Clicks.
CASE (PushButton)

! Branch on the buttons on the dialog.
SELECT CASE (MESSAGE%VALUE1)

! Add To Grid Button.
CASE (ID_MAKE_RUNOFF_GRP)
CALL WDialogSelect(IDD_GROUP_RUNOFF_MODULES)

! increment number of rows in the CAL_GRID by using an invisible label on the dialog
if (no_make_row > 0) then
CALL WDialogGetInteger(IDF_LABEL_norows, no_make_row)
endif
if (no_make_row < 0) then
no_make_row = 0 ! This is not good programming, discuss with Grant
endif

```

```

no_make_row = no_make_row + 1
CALL WDialogPutInteger(IDF_LABEL_norows, no_make_row)
! make hidden
!CALL WDialogFieldState(IDF_LABEL_norows, 3)
CALL EH_GroupRunOffModules_Add(ITYPE,MESSAGE)

! Delete From Grid Button.
CASE (ID_DELETE_RUNOFF_GRP)
CALL EH_GroupRunOffModules_Delete(ITYPE,MESSAGE)

CALL WDialogGetInteger(IDF_LABEL_norows, no_make_row)
if (no_make_row > 0) then
  no_make_row = no_make_row - 1
end if
CALL WDialogPutInteger(IDF_LABEL_norows, no_make_row)

! Apply To All Runoff Modules Grid Button.
CASE (ID_LABEL3_CHOOOSE_ALL)
CALL EH_GroupRunoffModules_ApplyAll(ITYPE,MESSAGE)

! Apply to Some Grouped Runoff Modules Button.
CASE (ID_LABEL4_CHOOOSE_SELECTED)
CALL EH_GroupRunOffModules_ApplySome(ITYPE,MESSAGE)

! OK Grid Button.
CASE (ID_OK)
CALL EH_GroupRunOffModules_OK(ITYPE,MESSAGE)

! Cancel Button.
CASE (ID_CANCEL_RUNOFF_GRP)
CALL EH_GroupRunOffModules_Cancel(ITYPE,MESSAGE)

! Ignore all other buttons.
CASE DEFAULT

! End of buttons on the Run Off Edit Dialog.
END SELECT

! Ignore events from all other types of screen controls.
CASE DEFAULT

! End of events for dialog.
END SELECT

! End of dialog handle.
END IF

```

An important issue was to use an invisible label to keep track of the number of rows in the grid.

Finally the event handlers subroutines were as follows :

```

! =====
Subroutine EH_GroupRunOffModules_Show(ITYPE,MESSAGE)

```

```

!
! Hides the current dialog and shows the group run off dialog.
!
! =====
use resource
use winteracter
use library_module
implicit none

INTEGER :: ITYPE
TYPE(WIN_MESSAGE) :: MESSAGE
INTEGER :: r_mod, mnumber, i, NRU
INTEGER, PRIVATE :: fresult
LOGICAL :: OK
CHARACTER(LEN=5) :: rumods(1000)

! Create the array of available run off modules.
CALL WDialogLoad(IDD_GROUP_RUNOFF_MODULES)
mnumber = RU_HowMany()
!mnumber = 5
if( mnumber > 0 )then
! Extract all the modules of the type into the mods array.
do i = 1, mnumber
rumods(i) = RU_GetSortedRS(i)
end do
end if

! Bring up window to choose runoffs
! CALL WDialogHide()
CALL WDialogShow(-1,-1,0, Modeless)
CALL WDialogSelect(IDF_RUNOFF_CAL_MENU)
CALL WDialogPutMenu(IDF_RUNOFF_CAL_MENU, rumods, 5, 0)
end Subroutine

! =====
Subroutine EH_GroupRunOffModules_Add(ITYPE,MESSAGE)
!
! Adds selected run off module to the grid.
!
! =====
use resource
use winteracter
use library_module
use estring_module
implicit none

INTEGER :: ITYPE
TYPE(WIN_MESSAGE) :: MESSAGE
INTEGER :: r_mod, rmod, row_no
INTEGER, PRIVATE :: fresult, i, irow
LOGICAL :: OK
CHARACTER(LEN=5) :: rumods(1000)
CHARACTER(LEN=5) :: r_mod_a

```

```
CHARACTER(LEN=5) :: temp_rumods(1000), cvalue
```

```
OK = .true.
```

```
CALL WDialogSelect(IDF_RUNOFF_CAL_GRID)  
!CALL WGridRows(IDF_RUNOFF_CAL_GRID, 4)  
CALL WGridState(IDF_RUNOFF_CAL_GRID, 1, Enabled)
```

```
CALL WDialogSelect(IDD_GROUP_RUNOFF_MODULES)  
! Get the chosen module from the menu and convert to a string.  
CALL WDialogGetMenu(IDF_RUNOFF_CAL_MENU, r_mod)
```

```
CALL NTOA(r_mod, r_mod_a)
```

```
! Add the run off module to the bottom of the grid.
```

```
! Read how many runoff modules have been chosen from the invisible label on the dialog
```

```
CALL WDialogGetInteger(IDF_LABEL_norows, row_no)
```

```
! Check that it is not there already (in the grid)
```

```
do irow = 1,row_no  
  CALL WGridGetCellString(IDF_RUNOFF_CAL_GRID,1,irow,cvalue)  
  temp_rumods(irow) = cvalue  
end do
```

```
do i = 1,row_no
```

```
  if (r_mod_a .EQ. temp_rumods(i)) then
```

```
    OK = .false.
```

```
  end if
```

```
end do
```

```
if (OK) then
```

```
  CALL WGridPutCellString(IDF_RUNOFF_CAL_GRID, 1, row_no, r_mod_a)
```

```
else
```

```
  fresult = ES_Clear()
```

```
  fresult = ES_Add('This runoff module is already in the group ',0)
```

```
  CALL ES_Tell('Adding Runoff Module to Group')
```

```
  if (row_no > 0) then
```

```
    row_no = row_no -1
```

```
  endif
```

```
  CALL WDialogPutInteger(IDF_LABEL_norows, row_no)
```

```
endif
```

```
end Subroutine
```

```
! =====
```

```
Subroutine EH_GroupRunOffModules_Delete(ITYPE,MESSAGE)
```

```
  use resource
```

```
  use winteracter
```

```
  use estring_module
```

```
  implicit none
```

```
  INTEGER :: ITYPE
```

```
  TYPE(WIN_MESSAGE) :: MESSAGE
```

```
INTEGER, PRIVATE :: row_no, iFromRow, fresult, ncol, irow, i, j
CHARACTER(LEN=5) :: temp_rumods(1000), cvalue
```

```
! Read no of rows from invisible label to decrement
CALL WDialogSelect(IDD_GROUP_RUNOFF_MODULES)
CALL WDialogGetInteger(IDF_LABEL_norows, row_no)
```

```
CALL WDialogSelect(IDF_RUNOFF_CAL_GRID)
iFromRow = WinfoGrid(IDF_RUNOFF_CAL_GRID,5)
```

```
do irow = 1, WinfoGrid(IDF_RUNOFF_CAL_GRID,3)
  IF (WinfoGridCell(IDF_RUNOFF_CAL_GRID,1,irow,1).EQ.1) then
    CALL WGridGetCellString(IDF_RUNOFF_CAL_GRID,1,irow,cvalue)
    temp_rumods(irow) = cvalue
  end if
end do
temp_rumods(iFromRow) = ''
```

```
do irow = 1, WinfoGrid(IDF_RUNOFF_CAL_GRID,3)
  Call WGridClearCell(IDF_RUNOFF_CAL_GRID,1,irow)
end do
!row_no = WinfoGrid(IDF_RUNOFF_CAL_GRID,3) Doesn't work – returns 20 the design number of rows
j = 0
if (row_no > 1) then
  do i = 1, row_no
    if (temp_rumods(i) .NE. ' ') then
      j = j + 1
      CALL WGridPutCellString(IDF_RUNOFF_CAL_GRID, 1, j, temp_rumods(i))
    end if
  end do
end if
```

end Subroutine

```
! =====
```

```
Subroutine EH_GroupRunoffModules_ApplyAll(ITYPE,MESSAGE)
```

```
  use resource
  use winteracter
  use estring_module
  implicit none
```

```
  INTEGER :: ITYPE
  TYPE(WIN_MESSAGE) :: MESSAGE
  INTEGER, PRIVATE :: fresult, i
  INTEGER, PRIVATE :: nc
  CHARACTER(LEN=5) :: rumods(1000)
```

```
! The user pressed the APPLY to All Runoff Modules button
! Save the stuff, but don't close the dialog
do i = 1, NRU
```

```

nc = RU_GetIndex(i)

fresult = RU_EDIT_GS(nc,2)
if( fresult > 0 )then
  ! Set the active cursor button to 'Check'
!   CALL WDialogSetField(IDF_CHECK)
  end if
  ! Save the parameters in the proper array.
  if( RU_SaveParameters(nc) > 0 )then
    end if

  ! Set the values to those in the memory
  fresult = RU_EDIT_GS(nc,1)
end do
end Subroutine

! =====
Subroutine EH_GroupRunOffModules_OK(ITYPE,MESSAGE)
  use resource
  use winteracter
  implicit none

  INTEGER :: ITYPE
  TYPE(WIN_MESSAGE) :: MESSAGE

  ! Same as Cancel button
  ! akb5july2011 Next line needed otherwise it unloads the runoff edit Dialog ?
  CALL WDialogSelect(IDD_GROUP_RUNOFF_MODULES)
  CALL WDialogUnLoad(IDD_GROUP_RUNOFF_MODULES)

end Subroutine

! =====
Subroutine EH_GroupRunOffModules_ApplySome(ITYPE,MESSAGE)
  use resource
  use winteracter
  use library_module
  use estring_module
  implicit none

  INTEGER :: ITYPE
  TYPE(WIN_MESSAGE) :: MESSAGE
  INTEGER, PRIVATE :: some_mods(1000),row_no, i, nc, fresult
  LOGICAL, PRIVATE :: OK
  CHARACTER(LEN=7) :: r_mod_a

  CALL WDialogSelect(IDD_GROUP_RUNOFF_MODULES)
  CALL WDialogGetInteger(IDF_LABEL_norows, row_no)
  do i = 1, row_no
    CALL WGridGetCellString(IDF_RUNOFF_CAL_GRID, 1, row_no, r_mod_a)
    CALL ATON(r_mod_a,some_mods(i), OK)
  end do

```

```

! The user pressed the APPLY to Some Runoff Modules button
! Save the stuff, but don't close the dialog
if (OK) then
  do i = 1, row_no
    nc = RU_GetIndex(some_mods(i))

    fresult = RU_EDIT_GS(nc,2)
    ! Save the parameters in the proper array.
    if( RU_SaveParameters(nc) > 0 )then
      end if

    ! Set the values to those in the memory
    fresult = RU_EDIT_GS(nc,1)
  end do
end if

end Subroutine

! =====
Subroutine EH_GroupRunOffModules_Cancel(ITYPE,MESSAGE)
  use resource
  use winteracter
  implicit none

  INTEGER :: ITYPE
  TYPE(WIN_MESSAGE) :: MESSAGE
  ! akb5july2011 Next line needed otherwise it unloads the runoff edit Dialog ?
  CALL WDialogSelect(IDD_GROUP_RUNOFF_MODULES)
  CALL WDialogUnLoad(IDD_GROUP_RUNOFF_MODULES)

end Subroutine

```

Note 1: The Save_parameters function is used to save all three sets of calibration parameters (Pitman, Sami and Hughes) which are stored in the arrays A_POW, A_ST, etc. irrespective of which mode you are using.

Note 2: The RU_EDIT_GS(nc,1) call is used to copy what is on the screen to the appropriate parameters

Note 3: The RU_EDIT_GS(nc,2) copies back what is in the appropriate parameters to the screen but after validation of data has been done

Note 4: The command WGridRows is useful to ascertain what row you are on in a grid

11.15 Rainfall files

There are numerous functions that are required with monthly and daily rainfall analysis as shown in the following schematic for daily rainfall.

WRSM2004

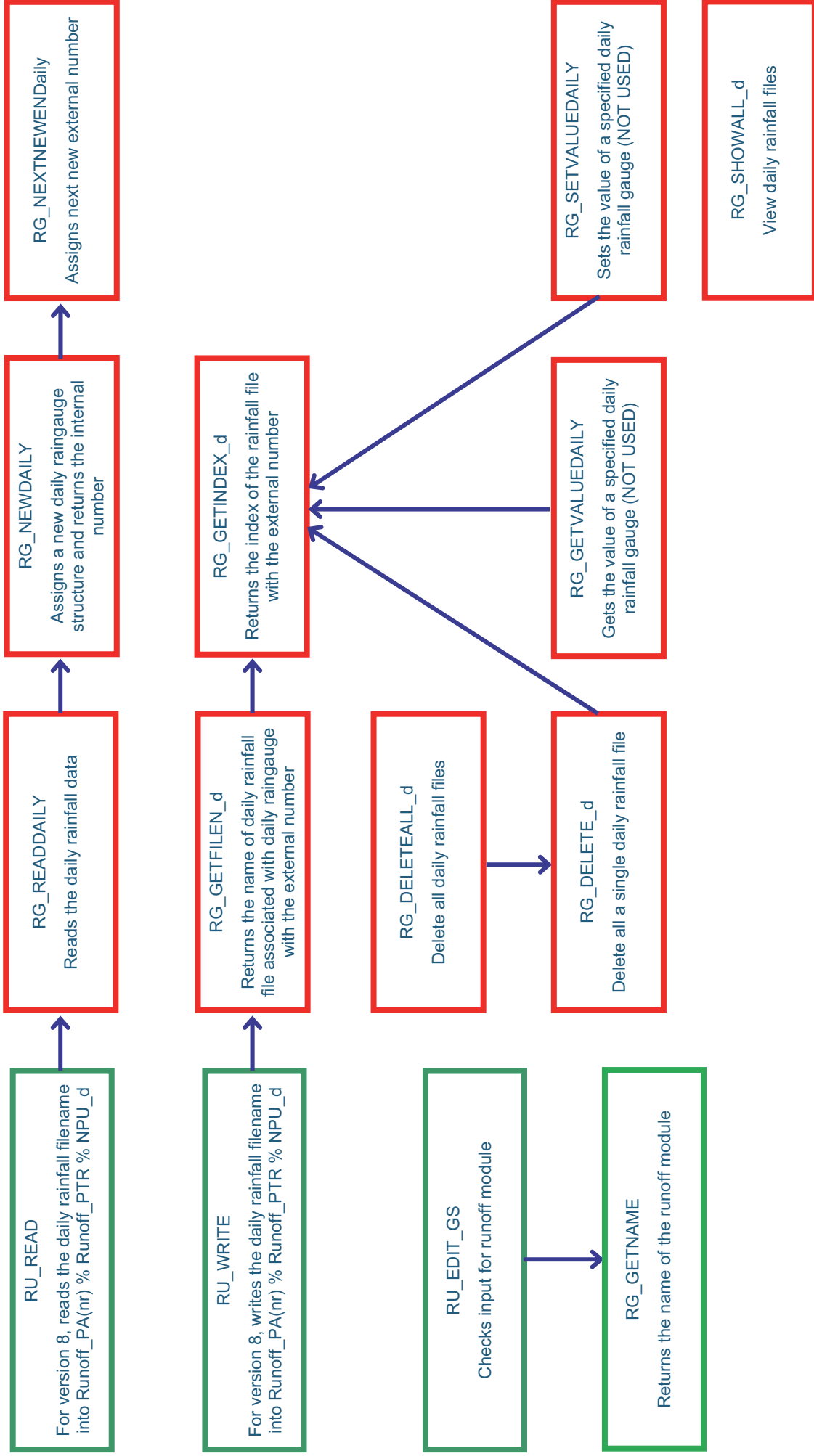


Figure 11-4: Daily Rainfall File Flowchart of Functions

11.16 Establishing a New Menu Item (off the Main Menu)

For example, adding a new menu item to the File menu such as the option to determine a daily land use file (say Create Daily). Go into the Resource.rc datafile and use the MenuEd editor. The File menu will automatically appear, put the cursor on Create Rainfile (or wherever you want) and go into Edit and click on Insert After. Then add Create Daily or whatever. Decide on a letter to represent this option (say the y in this case). Then use Create Daily (the ampersand before the y). Then under ID enter ID_FILE_CREATE_DAILY. This cannot be changed so make sure it is what you want. No need to have any of the 4 boxes ticked here (separator, Popup. etc.). Then Save. This will set up the following:

1) In the Resource.f90 file there will be one additional line as follows :

```
INTEGER, PARAMETER :: ID_FILE_CREATE_DAILY = 40121
```

(last line before END MODULE Resource at the end of the file)

Check that there is no other line 40121 (there was not in this case)

2) In the Resource.rc file, searching for ID_FILE_CREATE_DAILY will show two additions as follows :

```
# define ID_FILE_CREATE_DAILY 40121
```

And also

```
MENUITEM "Create Daily", ID_FILE_CREATE_DAILY
```

Now Compile All and Link and the new menu item will be visible.

Now it is up to the user to go into WRSM2001.f90 and using a CASE statement and subroutine CREATE_DAILY to set up code to action the required methodology.

11.17 Establishing a New Grid

For example, the daily time step grid for adding naturalised flow files.

To get the correct alignment of the one column, go into Modify and make the width a number that is 14 less than the width given in the main menu, so that the column does not overlap the grid boundary. For example for this daily time step grid the width (under Modify) is 184 whereas the actual grid is 198.

11.18 Establishing a Pop Up Menu Associated with the Plot Menu

For example : The daily time step including land use issue

The new dialog was first created.

Then in PLOTSPB in **wrsm2001.f90**, the CASE statements had to be revised to include option 12 – daily time step use including land use.

Then a subroutine was written to code the new pop up window called ShowDailyFiles

Then the following code was added, under the line

```

! Catch the messages that follow until the Cancel button is pressed.

CASE (FieldChanged)
  if((MESSAGE%VALUE1.EQ.      IDF_DTS_LU_RADIO).AND.      (MESSAGE%VALUE2.EQ.
IDF_DTS_LU_RADIO)) then
      CALL ShowDailyFiles(graph)
  end if

```

By trial and error it was found that both MESSAGE%VALUE1 and MESSAGE%VALUE2 had to be included in the if statement. They both give a value of 1504.

Note that Fieldchanged had to be trapped for.

IDF_DTS_LU_RADIO is the radio button for the daily time step including land use.

The graph structure had to be changed to add the two parameters for the simulated and observed files. This required the following command in the ShowDailyFiles subroutine :

```

type(Graph_Type), INTENT(IN OUT)  :: graph

```

Extending the graph structure was done in **wrsm2000.f90** – a part of the code is shown below –

```

module types_module
  ! The module types_module contains the types that can be used globally.
  !
  =====
  use constants_module
  implicit none
  ! The type graph type transfers data from the main program to the
  ! graph module class.
  type :: Graph_Type
  ....
  ! akb6feb2013 Add daily files to graph structure
  CHARACTER(LEN=300) :: filen1, filen2

```

In **wrsm2014.f90**, the following was required (i.e. graph % filen1):

```

OPEN(DLU_kIN6,FILE=graph % filen1,STATUS='OLD',ERR=2650)

```

11.19 Adjusting length of allowable text in module windows

For example increasing the length of dam name from 20 to 40 characters. Go into the Resource Editor via Dialogue Edit and establish the parameter for the name concerned, e.g. the reservoir name is IDD_RV_NAME. Then go back in choosing the Text Editor and search for this name. It comes up associated with the number 1175. Searching for this number will come up with 1175 20, change to 1175 40. Changes may also be required in the f90 file associated with reservoirs to extend a string.

11.20 Changing the Help/About Window

Go into Resource using the Text Editor and simply change text. For a new logo, change the SSI_Smaller.BMP to RHDHV_Smaller.bmp. Paint can be used to adjust the bmp file. If new lines are to be added then use the Dialog Editor.

For a new help item, go into resource.rc and choose the Menu Editor and add there. Give it a ID_... name which gets referred to in wrsm2001.f90.

11.21 Changing the Groundwater Abstractions in SAMI

Prior to October 201, groundwater abstractions were either off or had annual values. This was set by means of a tickbox. With this box ticked, a 1 was set via the logical parameter "AbstractionsOn" in the position highlighted in red below.

This is the "slbru23.dat" file in B20 system of the Olifants

```
23 'MU21: SAALBOOMSPRUIT' 7
268.4 654 'ZB2C.RAN'
3.0 0 67 5.0 3.2 999 999 1.5 0.40 2.50 0.5 1.00
183. 173. 190. 187. 156. 154. 118. 100. 81. 89. 117. 152.
0.80 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 0.80 0.80 0.80
0 0
0 0 0.00
0
2
46 15.0
45 85.0
0 0
2 0.000 3.000 2.000 2.000
3.0 0 67 5.0 0.0 999 999 1.5 0.40 0.00 0.5 1.00
1 0.40000
1 40.00 0.200 20.00 10.00 0.0010
0.0100
0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
31.77 0.0080 250.00 184.80 59.36
29.68 0.20 3.00 1.00 -0.05 25.00 2.00
0.00100000 10.00 1000.00 2.87
0.10 -3.00 0.00 8 1
3.0 0 77 6.0 0.0 999 999 1.5 0.25 0.00 0.5 1.00
8
1920 0.000
1950 0.000
1970 0.010
1995 0.020
1996 0.020
1998 0.050
2004 0.040
2009 0.040
0 2
3
4
0.65 0.81 0.81 0.81 0.80 0.80 0.77 0.75 0.72 0.59 0.62 0.64
```

This value will be a zero if there are no groundwater abstractions and the number of years (8 in this datafile) will be 0 as well. In the version set up in October 2014, this value can be a 0,1 or 2 being no groundwater, annual groundwater and a time series file of groundwater abstractions. This number is again assigned to the parameter "AbstractionsOn" but it is now in integer form to accommodate three options. Depending on its value it will expect certain data later on in the datafile. A bug was discovered in that some of the WR2005 datafiles had annual abstractions but this value was a 0. The old version read it OK as it acted on the number of years of abstraction data, however the latest version did not read correctly after the 0 (instead of a 1) and read the wrong values into A-pan evaporation and the model then gave an error on running.

12 FILE STRUCTURE

Module Input Parameter File Structure

The following is a discussion of the format of the input files for the different WRSM/Pitman modules. This should be of academic interest only to users of WRSM/Pitman who should rather use the built-in editing facilities of the program. It is not recommended that you edit the module data files outside of the program. This chapter is meant for programmers who want to write applications that need to read the files.

The figures in brackets are the format specifiers of the variables:

F – Floating point value (followed by the field length and the number of decimals)

I – Integer value (followed by the field length)

C – Character string value (followed by the maximum size)

12.1 The Runoff module parameter file

The following is an example of a Runoff Module parameter file with the line type numbers in front of every line. Note that these line type numbers (L.xxx) do not appear in the files that are created by WRSM/Pitman.

```
L.001 1 'C2C Runoff 1' 4
L.002 1350.0 586 'C2C_rain.txt'
L.003 2.1 0 340 13.0 12.0 50 999 1.0 0.50 2.50 0.0 1.0
L.004 213. 219. 227. 220. 172. 156. 121. 101. 79. 94. 133. 178.
L.005 0.70 0.80 0.80 0.80 0.80 0.80 0.70 0.60 0.50 0.50 0.50 0.60
L.006 0 2
L.007 1920 2000
L.008 5 6
L.009 30 20 25 20 45 20
L.010 50
L.011 2 1
L.012 1920 2000
L.013 6 8
L.014 10 15 30 15 60 15
L.015 50
L.016 2
L.017 1920 2000
L.018 0.01 0.02
L.019 1
L.020 1 100.0
L.021 0 0
L.022 2 0.000 0.000 0.000
L.023 0.0 0 0 0.0 0.0 0 0 0.0 0.00 0.00 0.0
L.024 1 0.00000
L.025 1 40.00 0.0055 0.50 10.00 0.0105
L.026 1.0000
L.027 0.00
L.028 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
L.029 0.00
L.030 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
L.031 30.00 0.0270 801.00 780.00 0.80
L.032 0.20 2.80 2.00 5.00 -0.05
```

L.033 0.00000600 3000.00 20000.00 3.00
 L.034 40.00 -5.00 0
 L.035 2.0 0 330 12.0 13.0 50 999 1.0 0.50 2.50 0.0 1.0
 L.036 7
 L.037 1900 24.00
 L.037 1985 24.00
 L.037 1986 0.10
 L.037 1987 9.84
 L.037 1988 21.96
 L.037 1989 25.20
 L.037 2000 25.20
 L.038 1 1
 L.039 10
 L.040 0.640 0.790 0.800 0.800 0.790 0.790 0.760 0.730 0.700 0.570 0.600 0.630

Note : If there were no Children then the last row would be 39

Line Type 1	Variable 1	The module number of this module (I4)
	Variable2	The name of this runoff module (in apostrophes) (C20)
	Variable 3	The file version number with which this file was written
		If this variable is absent, version 0 is assumed. The version number of the data file is important and is used when reading the file. When no special mention of the version number is made, version 0 is assumed. (I2)
Line Type 2	Variable 1	The catchment area (F7.1)
	Variable 2	The MAP of the catchment. (I4)
	Variable 3	The rainfall file name in apostrophes. Name only, the file is assumed to be in the network input directory. (C100)
Line Type 3	Variable 1	POW for the Pitman Model (F3.1)
	Variable 2	SL for the Pitman Model (I5)
	Variable 3	ST for the Pitman Model (I5)
	Variable 4	FT for the Pitman Model (F4.1)
	Variable 5	GW for the Pitman Model (F4.1)
	Variable 6	ZMIN for the Pitman Model (I5)
	Variable 7	ZMAX for the Pitman Model (I5)
	Variable 8	PI for the Pitman Model (F4.1)
	Variable 9	TL for the Pitman Model (F4.2)
	Variable 10	GL for the Pitman Model (F5.2)

	Variable 11	R for the Pitman Model (F3.1)
	Variable 12	FF (F5.2)
Line Type 4	Variable 1-12	Monthly Symons Pan evaporation values, starting in October. (F4.0)
Line Type 5	Variable 1-12	Monthly pan factors, starting in October (F4.2)
Line Type 6	Variable 1	The number of years for which there is afforestation data (NYRF) (I4)
	Variable 2	Variable 2 (FORESTALG) will only be read if the file version number is greater than 0. For files with a file version number of 0, the Van der Zel algorithm will always be used.
		The afforestation algorithm to use:
		0 – Van der Zel algorithm
		1 – CSIR algorithm
		2 – Smoothed Gush/Pitman
		3 – LUT Gush
		4 – User defined reduction (I4)
Line Types 7 to 10 are only present if the number of years for which there is afforestation data (NYRF) is greater than 0.		
Line Type 7	Variables	NYRF Year values (integer) for all the years for which there is afforestation data. (I4)
Line Type 8	Variables	NYRF Afforestation area in km ² (F6.1)
Line Types 9 and 10 will only be present if NYRF is greater than 0 and file version number is greater than 0		
Line Type 9	Variable 1	Percentages of area under Pine (F6.2)
	Variable 2	Crop rotation period for Pine (I3)
	Variable 3	Percentages of area under Eucalyptus (F6.2)
	Variable 4	Crop rotation period for Eucalyptus (I3)
	Variable 5	Percentage area under Wattle (F6.2)
	Variable 6	Crop rotation period for Wattle (I3)
Line Type 10	Variable 1	Percentage of forest area considered to be optimal (F7.2)
	Variable 2	If the module is a “Child” and it has user defined reductions, then the following two parameters will be required: User defined SFR reduction for MAR (F7.2)
	Variable 3	User defined SFR reduction for low flows (F7.2)

Line Type 11 (and, when necessary, Line Types 12 to 15) will only be read if the file version number is greater than 0

Line Type 11	Variable 1	The number of years for which there is Alien Vegetation data (NYRA) (I4)
	Variable 2	Vegetation algorithm (VEGETATIONALG) The alien vegetation algorithm to use: 0 – None 1 – CSIR algorithm 2 – Riparian (I4)
	Variable 3	Area of catchment covered by riparian vegetation (F7.2)

Line Types 12 to 15 are only present if the number of years for which there is Alien Vegetation data (NYRA) is greater than 0.

Line Type 12	Variable	NYRA Year values to describe the alien vegetation area (I4)
Line Type 13	Variables	NYRA Area values to describe the alien vegetation. (F6.1)
Line Type 14	Variable 1	Percentage area Tall Trees (F7.2)
	Variable 2	Age of Tall Trees (F7.2)
	Variable 3	Percentage area Medium Trees (F7.2)
	Variable 4	Age of Medium Trees (F7.2)
	Variable 5	Percentage area Tall Shrubs (F7.2)
	Variable 6	Age of Tall Shrubs (F7.2)
Line Type 15	Variable 1	percentage of alien vegetation area considered to be optimal (F6.2)

Line Type 16	Variable 1	The number of years for which there is Paved area data (NYRP) (I4)
--------------	------------	--

Line Types 17 and 18 are only present if the number of years for which there is paved area data (NYRP) is greater than 0.

Line Type 17	Variable	NYRP Year values to describe the paved area (I4)
Line Type 18	Variables	NYRP Values to describe the paved area as proportion of the total catchment. (F6.3)

Line Type 19	Variable 1	The number of outflow routes to the Runoff Module (I3)
--------------	------------	--

Line Type 20 is repeated for every outflow route to the Runoff Module:

Line Type 20	Variable 1	The outflow route number (I4)
	Variable 2	The percentage of the total outflow from the catchment that is routed via this outflow route. (F5.1)

The following lines will only be read if the file version number is greater than 2

The Hughes groundwater model parameters

Line Type 21	Variable 1	The inflow route number from upstream (I4)
	Variable 2	The number of the runoff module that influences the groundwater of this module. (I4)
Line Type 22	Variable 1	The groundwater model to use in this module: 0 – Classic Pitman Model 1 – Hughes Groundwater Model 2 – Sami Groundwater Model (I3)
	Variable 2	The Hughes Model Parameter HGSL (F8.3)
	Variable 3	The Hughes Model Parameter GPOW (F8.3)
	Variable 4	The Hughes Model Parameter TLGMax (F8.3)
	Variable 5	The Hughes Model Parameter HGGW (from file version 4 onwards, not in version 3) (F8.3)

Line Type 23 will only be present if the file version number is greater than version 3

Line Type 23	Variable 1	POW for the Hughes Model (F3.1)
	Variable 2	SL for the Hughes Model (I5)
	Variable 3	ST for the Hughes Model (I5)
	Variable 4	FT for the Hughes Model (F4.1)
	Variable 5	GW for the Hughes Model (F4.1)
	Variable 6	ZMIN for the Hughes Model (I5)
	Variable 7	ZMAX for the Hughes Model (I5)
	Variable 8	PI for the Hughes Model (F4.1)
	Variable 9	TL for the Hughes Model (F4.2)
	Variable 10	GL for the Hughes Model (F5.2)
	Variable 11	R for the Hughes Model (F3.1)

Variable 12 FF (F5.2)

Line Type 24 will only be present if the **file version number** is greater than 3

Line Type 24 Variable 1 Use number of reaches to calculate the drainage density?

1 = Yes

0 = No (I3)

Variable 2 Drainage density. (F8.5)

Line Type 25 Variable 1 Number of reaches in the catchment (used to calculate the drainage density) (I4)

Variable 2 Riparian area width as percentage of the total width (F6.2)

Variable 3 Riparian strip factor (F7.3)

Variable 4 Rest water level (F6.2)

Variable 5 Transmissivity (F6.2)

Variable 6 Storativity (F6.4)

Line Type 26 Variable 1 The initial value for the groundwater slope for the upper AND riparian zone. (F8.4)

Line Type 27 Variable 1 The annual GW abstraction from the upper zone (F8.2)

Line Type 28 Variables 12 monthly abstraction demands (expressed as %) for the upper zone. (F6.2)

Line Type 29 Variable The annual abstraction from the riparian zone (F8.2)

Line Type 30 Variables 12 monthly abstraction demands (expressed as %) for the riparian zone. (F6.2)

The following lines will only be present if the file version number is greater than 3

Data for the Sami GW model.

Line Type 31 Variable 1 Aquifer thickness (F6.2)

Variable 2 Storativity (F6.4)

Variable 3 Initial aquifer storage (F6.2)

Variable 4 The static water level (F6.2)

Variable 5 Unsaturated storage (F6.2)

Line Type 32 Variable 1 Initial storage of the unsaturated zone (F7.2)

Variable 2 Percolation power (PPOW) (F7.2)

	Variable 3	GPOW (F7.2)
	Variable 4	Max Discharge (F7.2)
	Variable 5	The power of the surface water/groundwater interaction curve (F7.2)
	Variable 6	HGSL (F7.2)
	Variable 7	HGGW (F7.2)
Line Type 33	Variable 1	Maximum Hydrological gradient (F10.8)
	Variable 2	Transmissivity (F8.2)
	Variable 3	Borehole distance to the river (F8.2)
	Variable 4	Groundwater evaporation area. (F8.2)
Line Type 34	Variable 1	K2 (F7.2)
	Variable 2	K3 (F7.2)
	Variable 3	Interflow lag (F7.2)
	Variable 4	Number of months over which recharge must be averaged (F7.2)
	Variable 5	Use abstractions in the simulation? (I4) (Last setting before the module was saved) 0 = No 1 = Yes.
Line Type 35	Variable 1	POW for the Sami Model (F3.1)
	Variable 2	SL for the Sami Model (I5)
	Variable 3	ST for the Sami Model (I5)
	Variable 4	FT for the Sami Model (F4.1)
	Variable 5	GW for the Sami Model (F4.1)
	Variable 6	ZMIN for the Sami Model (I5)
	Variable 7	ZMAX for the Sami Model (I5)
	Variable 8	PI for the Sami Model (F4.1)
	Variable 9	TL for the Sami Model (F4.2)
	Variable 10	GL for the Sami Model (F5.2)
	Variable 11	R for the Sami Model (F3.1)

	Variable 12	FF (F5.2)
Line Type 36	Variable 1	The number of year/ groundwater abstractions pairs.(NYGA) (I4)
Line Type 37 will only be present if the number of year/ groundwater abstraction pairs (NYGA) is greater than 0. Line Type 37 will be repeated NYGA times.		
Line Type 37	Variable 1	Year (I4)
	Variable 2	Abstraction value (F7.3)
Line type 38 is independent of whether a groundwater model is used or not.		
Line Type 38	Variable 1	Produce Naturalised flows during the next simulation? (Last setting before the module was saved) (I4)
		1 = Yes, 0 = No.
	Variable 2	Number of Children (I4)
Line Type 39	Variable 1	If there are children, their module numbers (nI4)
Line Type 40	Variable 1	A-pan factors 12(IF4.2)

12.2 The channel reach submodel parameter file

The following is an example of a channel reach submodel parameter file.

The file has the name TSCR3.DAT, which means that it is a part of network TS, it is a channel reach submodel file, and the channel reach submodel number is 3.

```

L.001 3 'River 3' 2
L.002 0 'b'
L.003 0 1 1 1
L.004 1 1 1 1 1 1 1 1 1 1 1
L.005 1 1 1 1 1 1 1 1 1 1 1
L.006 6
L.007 4
L.008 1 'b'
L.008 3 'b'
L.008 4 'b'
L.008 11 'A2M42EFF.FLO'
L.009 2
L.010 6 'b'
L.010 5 'b'
L.011 1 2
L.012 0.000 0.000 0.000 0.000 0.000 0.000
L.013 0.000
L.014 4
L.015 1
L.016 2
L.017 2.000 0.500 0.100

```

Line type 1	Variable 1	Submodel number.
	Variable 2	Submodel name, maximum 20 characters. Note the quotes. There must be a blank (b) between the quotes if no name is given.
	Variable 3	File version number (Should be a 2 for networks generated with the version of WRSM/Pitman distributed in January 2006)
Line type 2	Variable 1	MAP for the wetland area associated with this channel reach – enter 0 if there are no wetlands.
	Variable 2	The name of the file with the rainfall as percentage of MAP. In this case the file name can be blank because there are no wetlands associated with the channel reach.
Line type 3	Variable 1	CHBL Monthly bed loss in channel (106m ³ /month)
	Variable 2	CHC1 Wetland/aquifer storage (million cubic meters)
	Variable 3	CHC2 Wetland/aquifer area (square km)
	Variable 4	CHC3 Wetland/aquifer recharge coefficient. (See notes on page 52)
Line type 4	Variable 1 to 12	Monthly pan evaporation values, starting at October (in mm). The type of pan used is left to the user. Together with the values in line type 5, the result should be wetland evaporation. (See notes on page 52)
Line type 5	Variable 1 to 12	Monthly evaporation pan factors, starting at October. (See notes on page 52)
Line type 6	Variable 1	The route number of the principal outflow route.
Line type 7	Variable 1	The number of inflow routes to this submodel.
Repeat line type 8 for every inflow route into the submodel.		
Line type 8	Variable 1	The inflow route number.
	Variable 2	The name of the file that contains defined flows for this route. Leave this name blank if the model is to simulate the flows in the route. Note the quotes, and the b to denote a blank or space.
Line type 9	Variable 1	The number of outflow routes that exit this submodel.
Repeat line type 10 for every outflow route into the submodel.		
Line type 10	Variable 1	The outflow route number.
	Variable 2	The name of the file that contains defined flows for this route. Leave this name blank if the model is to simulate the flows in the route. Note the quotes. It is not strictly necessary to specify the principal

outflow route.

Line type 11	Variable 1	Reserved for possible future use. Should always be a 1.
	Variable 2	A "0" for no wetland, a "1" for a basic wetland, a "2" for a comprehensive wetland and a 3 for a diversion channel.

Note : If a comprehensive channel has been chosen the following line can have the following :

Line type 12	Variable 1	the area of wetland at bankfill level of channel
	Variable 2	the volume of wetland at bankfill level
	Variable 3	the power of the area-volume relationship
	Variable 4	the bankfill capacity of river channel
	Variable 5	the proportion of flow in excess of bankfill into wetland
	Variable 6	the proportion of wetland volume (over bankfill) into channel.
Line type 13	Variable 1	A factor called QDIV which is a reserved parameter – not in use at present. This value should always be 0.0
Line type 14	Variable 1	Local wetlands inflow route
Line type 15	Variable 1	Local wetlands outflow/abstraction route number
Line type 16	Variable 1	Diversion outflow route number
Line type 17	Variable 1	Bankfill capacity of river channel
	Variable 2	Diversion efficiency
	Variable 3	Maximum monthly diversion capacity

NOTES:

1. Variables 2, 3 and 4 in line type 3 and all the variables in line types 4 and 5 are not used as there is no wetland. The program does, however, expect to read in these variables and that is why dummy values must be entered.
2. The variables CHC1 and CHC2 respectively refer to the nominal storage and area of the wetland/aquifer when full. However it is possible for the nominal wetland/aquifer capacity to be exceeded during the passage of a flood. The recharge coefficient CHC3 relates specifically to aquifers that are limited by the transmissivity of the alluvium. In all other cases, the value of CHC3 should be set equal to 1.

The potential recharge in any given month is given by the product of CHC3 and the available storage (i.e. CHC1 – (the current storage)) in the aquifer, plus a quantity related to the flow in the channel. Typical values for CHC3 for an aquifer would be in the order of 0.1.

12.3 Reservoir submodel parameter file

The following file has the name TSRV2.DAT, which means that it is a submodel parameter file which is a submodel to network TS, it is a Reservoir submodel, and the reservoir module number is 2.

```
L.001      2 'Reservoir 2'
L.002      604. 'MPA2M42.RAN' 1.
L.003      2
L.004      1920 1990
L.005      10 12
L.006      10 12
L.007      196 196 202 190 156 145 108 86 68 79 115 159
L.008      1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
L.009      3
L.010      1
L.011      2 'b'
L.012      2
L.013      12 'b'
L.013      3 'b'
```

Line type 1	Variable 1	The submodel module number.
	Variable 2	The submodel name. Note the parentheses.
Line type 2	Variable 1	The Mean Annual Precipitation (in mm).
	Variable 2	The name of the file with the rainfall as percentage of MAP. (Note the quotes).
	Variable 3	The volume/surface area power P, in the equation surface area = k * (volume) ^p .
Line type 3	Variable 1	The number of years for which there is volume and surface area data.
Line type 4	Variable 1	to the number of years with data: The years for which there is volume and area data.
Line type 5	Variable 1	to the number of years with data: The full supply volume (106m ³) of the reservoir in that given year.
Line type 6	Variable 1	to the number of years with data: The surface area of the reservoir at full supply volume (km ²).
Line type 7	Variable 1 to 12	Monthly pan evaporation, starting at the value for October (in mm). The type of pan used is left to the user. Used in conjunction with line type 8, the result should be lake evaporation.
Line type 8	Variable 1 to 12	Monthly pan factor, starting at the value for October.
Line type 9	Variable 1	The route number of the spillage route.
Line type 10	Variable 1	The number of inflow routes to the reservoir.

Repeat line type 11 for every inflow route.

Line type 11	Variable 1	The route number of an inflow route to the reservoir.
	Variable 2	The name of a file with flow data for the route if the route has defined flow. Note the quotes. If the model is to calculate the flows, enter a blank file name.

Line type 12	Variable 1	The number of outflow routes from the reservoir.
--------------	------------	--

Repeat line type 13 for every outflow route.

Line type 13	Variable 1	The route number of an outflow route from the reservoir.
	Variable 2	The name of a file with flow data for the route if the route has defined flow. Note the quotes. If the model is to calculate the flows, enter a blank file name.

Please note :

When the name of an outflow file is set to '*' on Line type 13 variable 2, the specified route becomes a "monthly defined abstractions" route. Such routes accept an additional line type, Line type 14. Line type 14 contains 12 month-specific volumes that are to be abstracted via this route, the storage state (of the reservoir) above which the full abstraction must be taken and the reduction factor by which the demand will be reduced if the storage state of the reservoir at the beginning of the month is less than the minimum storage state. Confused? Example:

```
0013 42 '*'
0014 1.4 1.5 1.6 1.3 1.3 1.3 1.0 1.1 1.2 1.2 1.2 1.25 7.25 0.5
```

According to line type 13, route 42 is a monthly defined abstractions route.

In any October, abstractions from the reservoir will be 1.4 million m³, in a November 1.5 million m³ and so on and 1.25 million m³ will be abstracted in a September. That is: will be abstracted if the reservoir storage is above 7.25 million m³ at the beginning of the particular month. If the storage state is below 7.25 million m³, the demand will be reduced by half (0.5) to 0.7 million m³ for an October or 0.65 million m³ if the month is a November. In effect therefore, water restrictions can be imposed if there is too little water in the reservoir. If the reduction factor is 1.00, no restrictions are imposed when the dam level falls below 7.25 because (1.4 * 1.00) = 1.4. If the reduction factor is 0.0 then no water will be supplied because (1.4 * 0.00) = 0.00 . See also: Graphics and Deficit handling.

Line type 15	Variable 1	The initial storage state of the reservoir in million cubic metres. This line is optional. If this line type is absent or the value is 01.00, the initial storage state of the reservoir will be set to 50% of FSV (full supply volume)
--------------	------------	---

12.4 The Irrigation submodel parameter file

The file has the name TSRR4.DAT, which means that it is a part of network TS, it is an irrigation submodel file, and the irrigation block submodel number is 4.

For the original WRSM/Pitman irrigation method the following example is applicable:

```
L.001 4 'Irrigation 4 ' 32
```

L.002 1
 L.003 700 'mpa2m42.ran'
 L.004 2
 L.005 1970 1990
 L.006 10.00 10.00
 L.007 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 L.008 196 196 202 190 156 145 108 86 68 79 115 159
 L.009 0.54 1.69 0.71 0.68 0.63 0.57 0.62 0.54 0.49 0.63 0.66
 L.010 9999 1 1 1 1 1 1 1 1 1 1 1
 L.011 5
 L.012 7 10.0

Other default data pertaining to the WQT and WQT-SAPWAT method

Line type 1	Variable 1	The submodel module number.
	Variable 2	The submodel name. Note the quotes.
	Variable 3	The file version number (set up within WRSM/Pitman, a '3' describes a WRSM + WQT model type and 2 is the version number)
Line type 2	Variable 1	The last used model type – 1 for WRSM/Pitman, 2 for WQT and 3 for WQT-SAPWAT
Line type 3	Variable 1	The Mean Annual Precipitation.
	Variable 2	The name of the file with the rainfall as percentage of MAP. (Note the quotes).
Line type 4	Variable 1	The number of years for which there is irrigation area data.
Line type 5	Variable 1	to the number of years with data: The years for which there is irrigation area data.
Line type 6	Variable 1	to the number of years with data: The area under irrigation in that given year (in km ²).
Line type 7	Variable 1 to 12	Monthly PINDEX values, starting at the value for October (i.e. area in each month as proportion of total irrigation area).
Line type 8	Variable 1 to 12	Monthly pan evaporation, starting at the value for October. The type of pan (usually Class A) is left to the user. Multiplied by the value for line type 8, the result should be evapotranspiration (Et).
Line type 9	Variable 1 to 12	Monthly pan/crop factor applicable to A-pan data (see WR90 Appendix 3.3.2), starting at the value for October.
Line type 10	Variable 1	Maximum annual irrigation allocation (mm) (Pitman method).
	Variable 2 to 13	Monthly effective rainfall factors, starting at the value for October.
Line type 11	Variable 1	The route number of the abstraction (or inflow) route to the irrigation area.

Line type 12 Variable 1 The route number of the return flow (or outflow) route from the irrigation block. Enter a route number of 0 if there are no return flows.

 Variable 2 The percentage of the inflow to the irrigation block that is returned via the return flow route. Enter 0 if there are no return flows.

Other default data pertaining to the WQT and WQT-SAPWAT method

For the WQT type 2 or wqt/sapwat type irrigation option, the following example is described:

```
L.001      1 'irr1' 32
L.002      2
L.003      990.00 'B4A.RAN' 0 0
L.004      2
L.005      1920 1989
L.006      0.00 60.00
L.007      1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
L.008      90.00 90.00 90.00 90.00 90.00 90.00 90.00 90.00 90.00 90.00 90.00 90.00
L.009      1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
L.010      9999.000 0.9000 0.9000 0.9000 0.9000 0.9000 0.9000 0.9000 0.9000 0.9000 0.9000 0.9000
0.9000 0.9000
L.011      8
L.012      0 9.00
L.013      1 8.60 2 1 1
L.014      0.200 1
L.015      0.3000 0.0000 96.0000 6.0000 0.7500 0.1500 0.0000 0.0000 0.0000 0.0000
L.016      0.000 0.000 400.000 1000.000 250.000 250.000
L.017      0.400 0.400 0.400 0.400 0.400 0.400 0.400 0.400 0.400 0.400 0.400 0.400
L.018      1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
L.019      1
L.020      0.70 0.40 0.30 0.30 0.20 0.80 0.90 0.20 0.10 0.20 0.30 0.40 2.00
L.021      890.000 890.000
L.022      2 1
L.023      1920 1989
L.024      90.000 91.000
L.025      2 1
L.026      1920 1989
L.027      0.800 0.700
L.028      2 1
L.029      1920 1989
L.030      60.000 70.000
```

Line type 1 Variable 1 The external submodel module number.

 Variable 2 The submodel name. Note the quotes.

 Variable 3 The file version number (set up within WRSM/Pitman, a '3' describes a WRSM + WQT model type and 2 is the version number)

Line type 2 Variable 1 The last used model type – 1 for WRSM/Pitman, 2 for WQT and 3 for WQT-SAPWAT

Line type 3 Variable 1 The MAP

	Variable 2	The catchment based rainfall file
	Variable 3	A "0" if the drought reduction factor is not chosen and a "1" if it is (normally used with the WQT-SAPWAT) method
	Variable 4	A "1" if the allocation method is to be clip individual high months and a "0" if the default of reduce all months proportionally is used.
Line type 4	Variable 1	The number of years defining the area of irrigation
Line type 5	Variable 1	to the number of years with data. The years for which there is irrigation area data.
Line type 6	Variable 1	to the number of years with data. The area under irrigation in that given year (in km ²).
Line type 7	Variable 1 to 12	PIndex factors for each month, starting in October.
Line type 8	Variable 1 to 12	Monthly pan evaporation, starting at the value for October. The type of pan (usually Class A) is left to the user. Multiplied by the value for line type 7, the result should be evapotranspiration (Et).
Line type 9	Variable 1 to 12	Monthly pan/crop factors applicable to A-pan data (see WR90 Appendix 3.3.2),, starting at the value for October.
Line type 10	Variable 1	Maximum annual irrigation allocation (in mm) (Pitman method)
	Variable 2 to 13	Monthly effective rainfall factors, starting at the value for October.
Line type 11	Variable 1	Abstraction route number
Line type 12	Variable 1	Number of the return flow route
	Variable 2	The percentage of the abstracted flow that is return flow
Line type 13	Variable 1	Interpolation type for area (linear, exponential or defined)
	Variable 2	Maximum water allocation (MCM per yr) (WQT method)
	Variable 3	Number of data points for water allocation
	Variable 4	Interpolation type (linear, exponential or defined)
	Variable 5	The number of the Runoff Module in which the irrigation block lies
Line type 14	Variable 1	Transfer canal seepage – proportion of seepage that returns to the return flow
	Variable 2	Whether to produce Net return flows for this irrigation block
Line type 15	Variable 1	Transfer canal – proportion of flow loss
	Variable 2	Transfer canal – proportion of salt loss

	Variable 3	Irrigation Efficiency factor
	Variable 4	Return flow factor
	Variable 5	Proportion of return flow from upper zone
	Variable 6	Proportion of return flow from lower zone
	Variable 7	Deep percolation salt concentration factor
	Variable 8	Proportion of irrigated land salt loss
	Variable 9	Salt load applied to irrigated land 1
	Variable 10	Salt load applied to irrigated land 2
Line type 16	Variable 1	Initial salt load – upper zone
	Variable 2	Initial salt load – lower zone
	Variable 3	Soil moisture storage capacity – upper zone
	Variable 4	Soil moisture storage capacity – lower zone
	Variable 5	Soil moisture storage – target
	Variable 6	Soil moisture storage – initial
Line type 17	Variable 1 to 12	Monthly rainfall factors.
Line type 18	A-pan factors	
Line type 19	Number of crops	
Line type 20	Variable 1 to 12	Monthly crop factors one data set for each type of crop
	Variable 13	Crop percentage (CPF) for the crop
	Variable 14	Crop description (in single quotes)
Line type 21	Variable 1	Effective rainfall factor 1
	Variable 2	Effective rainfall factor 2
Line type 22	Variable 1	The number of years for irrigation allocation growth
	Variable 2	Interpolation type
Line type 23	Variable 1	to the number of years with data: The years for which there is irrigation allocation growth data.
Line type 24	Variable 1	to the number of years with data: irrigation allocation growth data in that given year
Line type 25	Variable 1	The number of years of return flow data

	Variable 2	Interpolation type
Line type 26	Variable 1	to the number of years with data: The years for which there is return flow data.
Line type 27	Variable 1	to the number of years with data: The return flow data for that given year.
Line type 28	Variable 1	The number of years of efficiency data
	Variable 2	Interpolation type
Line type 29	Variable 1	to the number of years with data: The years for which there is irrigation efficiency data.
Line type 30	Variable 1	to the number of years with data: The irrigation efficiency data for that given year.

For the WQT type 4 irrigation option, the following example is described:

Test file BRrr4.dat (Breede)

```

1  4 'Fdam Irrig G50D' 42
2  4
3  437.00 'g5a.ran' 0 0
4  9
5  1920 1942 1982 1983 1986 1989 2000 2004 2009
6  1.00 1.00 1.00 2.00 2.00 2.00 1.03 1.03 1.03
7  0.83 0.83 0.83 0.79 0.79 0.79 0.79 0.79 0.62 0.62 0.83 0.83
8  161.00 225.00 264.00 271.00 229.00 200.00 124.00 72.00 55.00 61.00 72.00 105.00
9  0.45 0.49 0.47 0.48 0.63 0.55 0.38 0.28 0.23 0.25 0.27 0.39
10 9999.000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
11 3
12 7 10.00
13 1 20.00 0 1 1
14 0.000 1

```

15 0.0000 0.0000 0.8500 0.0200 0.7500 0.1500 0.0000 0.0000 0.0000 0.0000
 16 0.000 0.000 400.000 1000.000 250.000 250.000
 17 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 18 0.580 0.610 0.610 0.610 0.600 0.600 0.570 0.460 0.360 0.290 0.360 0.480
 19 1
 20 0.450 0.490 0.470 0.480 0.630 0.550 0.380 0.280 0.230 0.250 0.270 0.390 100.00 'composite'
 21 100.000 0.000
 22 0 1
 23 not req
 24 not req
 25 3 1
 26 1920 2004 2009
 27 1.000 1.000 1.000
 28 2 1
 29 1920 2009
 30 1.000 1.000
 31 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 32 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600
 33 2 1
 34 1920 2009
 35 0.00 0.00
 36 0.000
 37 0.00
 38 400.00
 39 50.00
 40 0.000

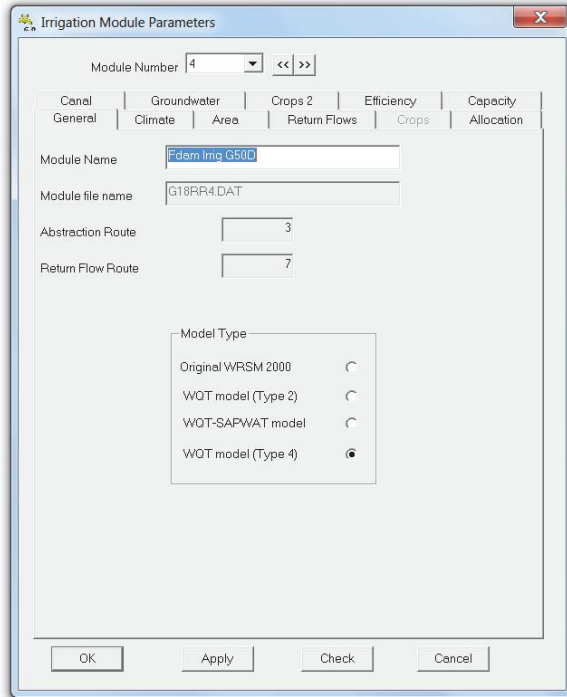


Figure 12-1: WQT Type 4: General Screen

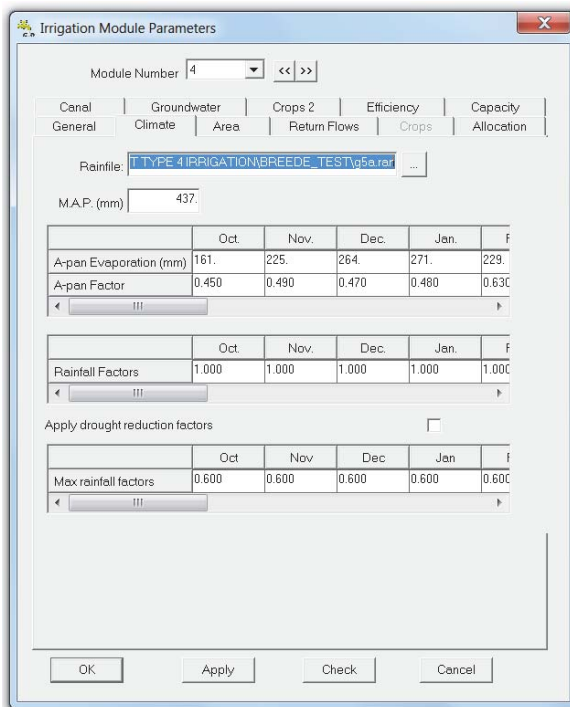


Figure 12-2: WQT Type 4: Climate Screen

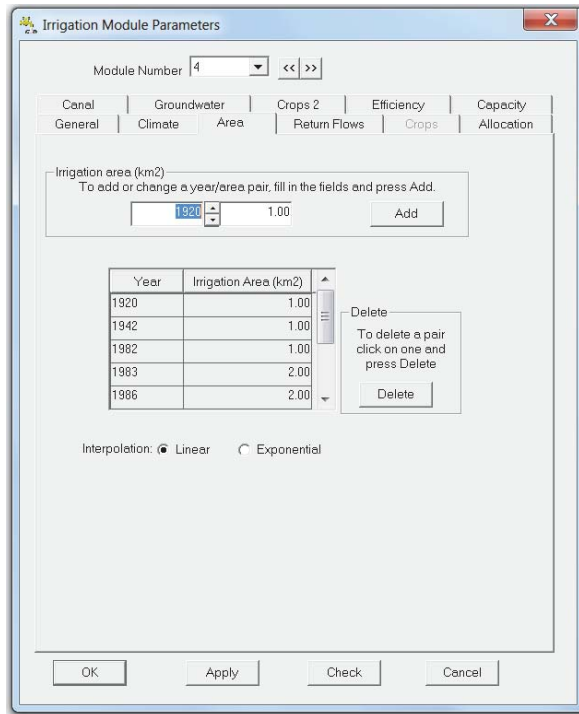


Figure 12-3: WQT Type 4: Area Screen

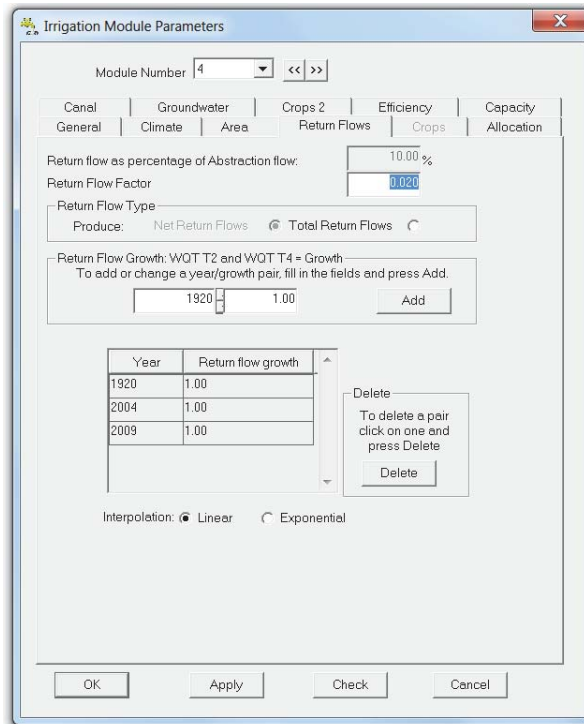


Figure 12-4: WQT Type 4: Return Flow Screen

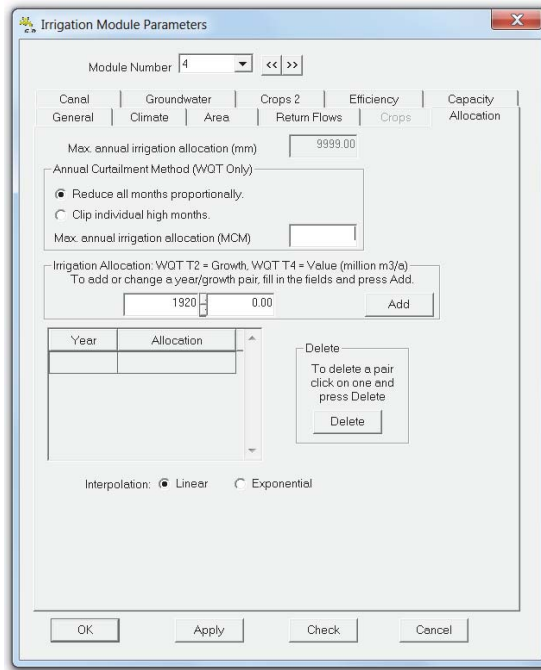


Figure 12-5: WQT Type 4: Allocation Screen

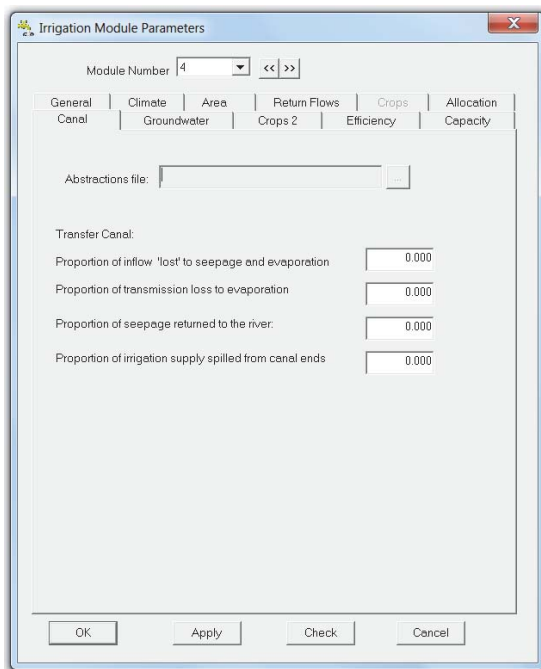


Figure 12-6: WQT Type 4: Canal Screen

Module Number 4 << >>

General | Climate | Area | Return Flows | Crops | Allocation
Canal | Groundwater | Crops 2 | Efficiency | Capacity

Module lies in Runoff Module 1 (G50D)

Soil Moisture Parameters

Soil Moisture Storage Capacity

Upper Zone (mm) 400.000

Lower Zone (mm) 1000.000

Soil Moisture Storage

Initial (mm) 250.000

Target (mm) 250.000

Proportion of Return Flow

Upper Zone 0.750

Lower Zone 0.150

Upper soil zone maximum storage (mm) 400.000

Upper soil zone minimum storage (mm) 50.000

Loss to deep-seated groundwater as proportion of irrigation inflow to lower soil zone 0.00

OK Apply Check Cancel

Figure 12-7: WQT Type 4: Groundwater Screen

Module Number 4 << >>

General | Climate | Area | Return Flows | Crops | Allocation
Canal | Groundwater | Crops 2 | Efficiency | Capacity

Fill in the Percentage area covered by up to 20 specific crops and the 12 monthly crop demand factors for each of these crops.

A specific crop will only be taken into account if the crop coverage is more than 0%.

	Crop Type	Percentage	Oct	Nov	De
1	composite	100.	0.45	0.49	0.47
2					
3					

Effective Rainfall Limit 1 (mm/month) 100.00

Effective Rainfall Limit 2 (mm/month) 0.00

Used in WQT-SAPWAT. Oct - Sep values represent crop requirements - evapotranspiration (only first row required)

OK Apply Check Cancel

Figure 12-8: WQT Type 4: Crops 2 Screen

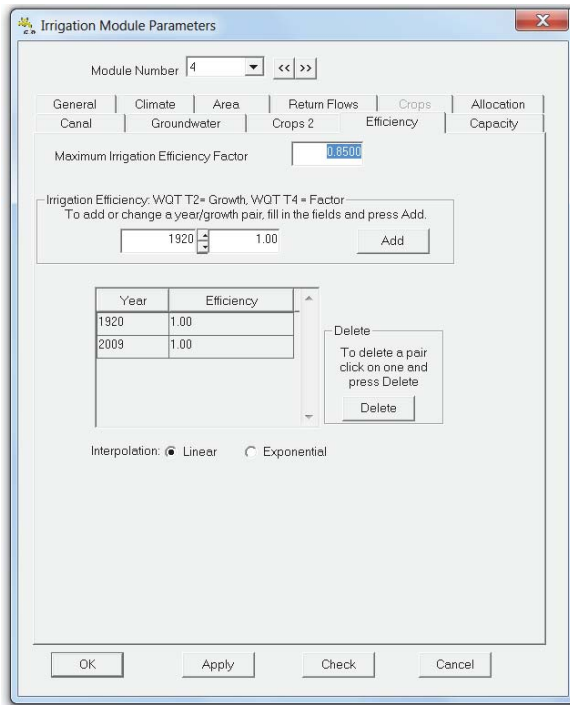


Figure 12-9: WQT Type 4: Efficiency

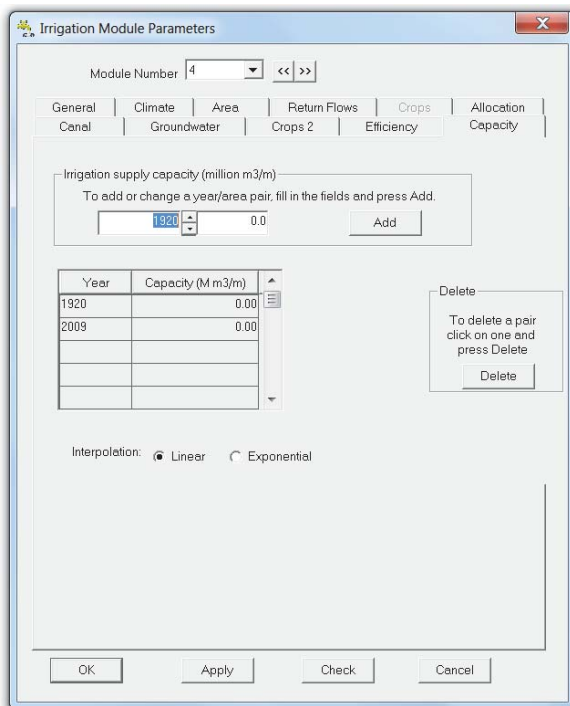


Figure 12-10: WQT Type 4: Capacity Screen

Red gives the parameter description, blue the parameter name if used in WQT Type 4 and green the parameter name if not used in WQT Type 4.

Line 1: 4 'Fdam Irrig G50D' 42

Module number, module name, file version

Irrig_PA(nc) % Irrig_PTR % NRREN, Irrig_PA(nc) % Irrig_PTR % RRNAM, CURRENTFILEVERSION

Line 2: 4

WQT Type 4

Irrig_PA(nc) % Irrig_PTR % MODEL

Line 3: 437.00 'g5a.ran' 0 0

MAP, MAP file name, drought reduction option, clipping option

Irrig_PA(nc) % Irrig_PTR % RRMAP, Irrig_PA(nc) % Irrig_PTR % NPI, Irrig_PA(nc) % Irrig_PTR % DROUGHT_APPLICABLE, Irrig_PA(nc) % Irrig_PTR % CurtailByClipping

Line 4: 9

Number of year pairs for irrigation area

Irrig_PA(nc) % Irrig_PTR % NYR

Line 5: 1920 1942 1982 1983 1986 1989 2000 2004 2009

Years

Irrig_PA(nc) % Irrig_PTR % ARR(i) % YEAR

Line 6: 1.00 1.00 1.00 2.00 2.00 2.00 1.03 1.03 1.03

Irrigation areas

Irrig_PA(nc) % Irrig_PTR % ARR(i) % VALUE

Line 7: 0.83 0.83 0.83 0.79 0.79 0.79 0.79 0.79 0.62 0.62 0.83 0.83 (Stored only for Pitman mode in Crops screen)

Pindex factors

Irrig_PA(nc) % Irrig_PTR % PINDEX(i)

Line 8: 161.00 225.00 264.00 271.00 229.00 200.00 124.00 72.00 55.00 61.00 72.00 105.00

A-pan evaporation

Irrig_PA(nc) % Irrig_PTR % APAN(i)

Line 9: 0.45 0.49 0.47 0.48 0.63 0.55 0.38 0.28 0.23 0.25 0.27 0.39

A-pan factors

Irrig_PA(nc) % Irrig_PTR % APF(i)

NB Comment: The input file has two sets of Apan factors.

Line 9 is used for WQT Type 4 whereas line 18 is used for WQT Type 2

Line 10: 9999.000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
1.0000 only for Pitman in Crops screen

Allocation, Crop factors

Irrig_PA(nc) % Irrig_PTR % PERM, Irrig_PA(nc) % Irrig_PTR % EFPF(i)

Line 11: 3

Abstraction route

Irrig_PA(nc) % Irrig_PTR % NRRARN

Line 12: 7 10.00

Return flow route, return flow %

Irrig_PA(nc) % Irrig_PTR % NRRRRN, Irrig_PA(nc) % Irrig_PTR % RRPERC

Line 13: 1 20.00 0 1 1

Interpolation type for area, maximum water allocation, number of data points, interpolation type for intent ?, associated runoff module

Irrig_PA(nc) % Irrig_PTR % KA + 1, Irrig_PA(nc) % Irrig_PTR % RRMA, Irrig_PA(nc) % Irrig_PTR % NPMA,
Irrig_PA(nc) % Irrig_PTR % KMA + 1, Irrig_PA(nc) % Irrig_PTR % IRRSW

Line 14: 0.000 1

Proportion of seepage to return flow, net return flow (1)/total return flow option (0)

Irrig_PA(nc) % Irrig_PTR % TLSQRF, Irrig_PA(nc) % Irrig_PTR % NETRETFLOWS

Comment: Net Return flows or Total return flows should give the same answer in Type 4

Line 15: 0.0000 0.0000 0.8500 0.0200 0.7500 0.1500 0.0000 0.0000 0.0000 0.0000

Transfer canal proportion of flow loss, transfer canal proportion of salt loss (not used), irrigation efficiency factor (gets assigned but does not appear to be used), return flow factor, proportion of return flow upper zone, proportion of return flow lower zone, deep percolation salt concentration factor (not used), proportion of irrigated land salt loss (not used), salt load applied to irrigation land 1 (not used), salt load applied to irrigation land 2 (not used),

Irrig_PA(nc) % Irrig_PTR % RRTLPQ, Irrig_PA(nc) % Irrig_PTR % RRTLPS, Irrig_PA(nc) % Irrig_PTR % RRIE,
Irrig_PA(nc) % Irrig_PTR % RRLF, Irrig_PA(nc) % Irrig_PTR % RRPRFU, Irrig_PA(nc) % Irrig_PTR % RRPRFL,
Irrig_PA(nc) % Irrig_PTR % RRSCF, Irrig_PA(nc) % Irrig_PTR % RRPSL, Irrig_PA(nc) % Irrig_PTR % RRSLD(1),
Irrig_PA(nc) % Irrig_PTR % RRSLD(2)

Line 16: 0.000 0.000 400.000 1000.000 250.000 250.000

Initial salt load upper zone (not used), initial salt load lower zone(not used), soil moisture capacity upper zone (not used), soil moisture capacity lower zone (not used), soil moisture target (not used), soil moisture storage initial

Irrig_PA(nc) % Irrig_PTR % RRSSUI, Irrig_PA(nc) % Irrig_PTR % RRSSLI, Irrig_PA(nc) % Irrig_PTR % RRHSU,
Irrig_PA(nc) % Irrig_PTR % RRHSL, Irrig_PA(nc) % Irrig_PTR % RRHT, Irrig_PA(nc) % Irrig_PTR % RRHI

Line 17: 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000

Effective rainfall factors

Irrig_PA(nc) % Irrig_PTR % RRERF(m)

Line 18: 0.580 0.610 0.610 0.610 0.600 0.600 0.570 0.460 0.360 0.290 0.360 0.480

Pan factors for A-pan evaporation

Irrig_PA(nc) % Irrig_PTR % APFWQ(m)

NB Comment: The input file has two sets of Apan factors.

Line 9 is used for WQT Type 4 whereas line 18 is used for WQT Type 2

Line 19: 1

Number of crops

Irrig_PA(nc) % Irrig_PTR % NCPS

Line 20: 0.450 0.490 0.470 0.480 0.630 0.550 0.380 0.280 0.230 0.250 0.270 0.390 100.00 'composite'

Crop factors, Crop Type description

Irrig_PA(nc) % Irrig_PTR % CF(i,m), Irrig_PA(nc) % Irrig_PTR % CROP_TYPE(i)

Line 21: 100.000 0.000

Effective rainfall factor 1, Effective rainfall factor 2

Irrig_PA(nc) % Irrig_PTR % RRERL1, Irrig_PA(nc) % Irrig_PTR % RRERL2

Line 22: 0 1

Number of points for allocation value (not growth), interpolation option

Irrig_PA(nc) % Irrig_PTR % NPMA, Irrig_PA(nc) % Irrig_PTR % KMA + 1

Line 23

Years for allocation value (not growth) (if applicable)

Irrig_PA(nc) % Irrig_PTR % WAG(i)

Line 24

Allocation value (not growth) (if applicable)

Irrig_PA(nc) % Irrig_PTR % WAG(i) % VALUE

Line 25: 3 1

Number of years for return flow, interpolation option

Irrig_PA(nc) % Irrig_PTR % NRFA, Irrig_PA(nc) % Irrig_PTR % KRFA + 1

Line 26: 1920 2004 2009

Return flow years

Irrig_PA(nc) % Irrig_PTR % RFG(i)

Line 27: 1.000 1.000 1.000

Growth in return flow factors

Irrig_PA(nc) % Irrig_PTR % RFG(i) % VALUE

Line 28: 2 1

Number of years for irrigation efficiency, interpolation option

Irrig_PA(nc) % Irrig_PTR % NIEG, Irrig_PA(nc) % Irrig_PTR % KIEG + 1

Line 29: 1920 2009

Irrigation efficiency years

Irrig_PA(nc) % Irrig_PTR % IEG(i) % YEAR

Line 30: 1.000 1.000

Irrigation efficiency value (not growth)

Irrig_PA(nc) % Irrig_PTR % IEG(i) % VALUE

Line 31: 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000

Minimum rainfall factors RRERFT4 (not used)

Irrig_PA(nc) % Irrig_PTR % RRERFT4(m)

Line 32: 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600 0.600

Maximum rainfall factors RRERFMT4

Irrig_PA(nc) % Irrig_PTR % RRERFMT4(m)

Line 33: 2 1

Number of years for capacity value, interpolation option

Irrig_PA(nc) % Irrig_PTR % NPCAP, Irrig_PA(nc) % Irrig_PTR % KCAP + 1

Line 34: 1920 2009

Capacity growth years

Irrig_PA(nc) % Irrig_PTR % NPCAP

Line 35: 0.00 0.00

Capacity growth factors

Irrig_PA(nc) % Irrig_PTR % RRCAPT4(m)

Comment: rcap set to above, does not have a value plus growth factors for different years, just values for different years

Line 36: 0.000

Canal transmission loss

Irrig_PA(nc) % Irrig_PTR % RRTLPE4

Line 37: 0.00

Deep seated groundwater

Irrig_PA(nc) % Irrig_PTR % RRPDLT4

Line 38: 400.00

Maximum allowed upper zone storage

Irrig_PA(nc) % Irrig_PTR % RRHMAXT4

Line 39: 50.00

Minimum allowed upper zone storage

Irrig_PA(nc) % Irrig_PTR % RRHMINT4

Line 40: 0.000

Irrigation supply spilled from canal ends

Irrig_PA(nc) % Irrig_PTR % RRPENDT4

12.5 The Mine submodel parameter file

```
L.001      1 'Mine1'  1
L.002      1  1  1  1
L.003      9 10
L.004     100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
L.005     0.81 0.82 0.83 0.84 0.88 0.88 0.88 0.87 0.85 0.83 0.81 0.81
L.006     'B4A.ran' 700.
L.007     0.44 0.70 0.00 0.00 0.00
L.008      2  1
L.009     1920 2003
L.010     0.31 0.31
L.011     'op'
L.012     20.000 3.000
L.013     1920 1 2003 2
L.014     0.450 0.440 0.250
L.015      2  1
L.016     1920 2003
L.017     0.00 0.23
L.018      2  1
L.019     1920 2003
L.020     0.00 0.20
L.021      2  1
L.022     1920 2003
L.023     0.00 0.40
L.024      2  6
L.025     1920 2003
L.026     0.00 0.30
L.027     0.80 0.70
L.028     1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
L.029     1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
L.030     0.000 0.00 0.00
L.031     1.90 0.60 0.01
L.032      2  1
L.033     1920 2003
L.034     0.00 0.25
L.035      2  1
L.036     1920 2003
L.037     0.00 0.40
L.038     0.010 0.20
L.039     0.19 0.50 0.01 0.00
L.040     0.00
L.041     0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
L.042     0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
L.043     0.00
```

L.044 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
 L.045 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
 L.046 'us' 10
 L.047 2.00 19.00 18.00 0.18
 L.048 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35
 L.049 0.10 0.10 0.10 0.10 0.10 0.10 0.10 0.10 0.10 0.10 0.10
 L.050 0.21 0.21 0.21 0.21 0.21 0.21 0.21 0.21 0.21 0.21 0.21
 L.051 2 1
 L.052 1920 2003
 L.053 0.00 0.77
 L.054 2 1
 L.055 1920 2003
 L.056 0.00 0.37
 L.057 0.00
 L.058 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
 L.059 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
 L.060 'dn'
 L.061 0.78 0.15 0.17 0.23 0.12 0.10 0.00
 L.062 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 L.063 2 1
 L.064 1920 2003
 L.065 0.00 0.29
 L.066 0.00
 L.067 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
 L.068 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000

Line Type 1	Variable 1	Mine module number.
	Variable 2	Name of mine module.
	Variable 3	File version.
Line Type 2	Variable 1	Runoff module in which the mining module lies
	Variable 2	Number of opencast sections
	Variable 3	Number of underground sections
	Variable 4	Number of slurry ponds
Line Type 3	Variable 1	Outflow route to river.
	Variable 2	Outflow route to central pollution control dam.
Line Type 4	Variable 1 to 12	Monthly evaporation values, starting at October (in mm).
Line Type 5	Variable 1 to 12	Monthly evaporation pan factors, starting at October.
Line Type 6	Variable 1	Rainfall file
	Variable 2	MAP (in mm)
Line Type 7	Variable 1	Plant area.
	Variable 2	Plant area runoff factor.

	Variable 3	Salt buildup rate (ton/km ² /month)
	Variable 4	Salt washoff efficiency calibration factor
	Variable 5	Initial salt store in the plant area (ton/km ²)
Line Type 8	Variable 1	Number of points for the growth of plant area
	Variable 2	the interpolation type.
Line Type 9	Variable 1	Start year for plant area growth factors.
	Variable 2	End year for plant area growth factors.
Line Type 10	Variable 1	Plant area growth factor for start year.
	Variable 2	Plant area growth factor for year end.
Line Type 11	Variable 1	Open cast section name.
Line Type 12	Variable 1	Area of coal reserves – km ²
	Variable 2	Workings area – km ²
Line Type 13	Variable 1	Commissioning year.
	Variable 2	Commissioning month.
	Variable 3	Decommissioning year.
	Variable 4	Decommissioning month.
Line Type 14	Variable 1	Disturbed area that contributes runoff to the pit/workings.
	Variable 2	Rehabilitated area with underlying inspoils aquifer – km ²
	Variable 3	Pit evaporation area.
Line Type 15	Variable 1	Number of points for growth or adjustment of workings area
	Variable 2	Interpolation option
Line Type 16	Variable 1	Years for pit/workings area
Line Type 17	Variable 1	Pit/workings area growth factors
Line Type 18	Variable 1	Number of points for growth of open cast disturbed area, runoff into workings
	Variable 2	Interpolation option.
Line Type 19	Variable 1	Years for growth factors of disturbed area.
Line Type 20	Variable 1	Growth factor of disturbed area that contributes runoff to the pit for specified years.

Line Type 21	Variable 1	Number of points for growth of open cast disturbed area, rehabilitated
	Variable 2	Interpolation option.
Line Type 22	Variable 1	Years for growth factors of rehabilitated area.
Line Type 23	Variable 1	Growth factor of rehabilitated area for specified years.
Line Type 24	Variable 1	Number of points for growth of open surface in pit evaporation area
Line Type 25	Variable 1	Years for growth factors for pit evaporation.
Line Type 26	Variable 1	Growth factors for pit evaporation.
Line Type 27	Variable 1	Runoff factor for disturbed area.
	Variable 2	Runoff factor for disturbed working area.
Line Type 28	Variable 1 to 12	Monthly recharge factors for disturbed area
Line Type 29	Variable 1 to 12	Monthly recharge factors for disturbed workings area
Line Type 30	Variable 1	Washoff efficiency calibration parameter
	Variable 2	Build-up rate of sulphate (ton/km ² /month)
	Variable 3	Initial mass of salts for simulation
Line Type 31	Variable 1	Inspoils storage where decant occurs
	Variable 2	Inspoils storage where seepage through weathered occurs.
	Variable 3	Initial inspoils storage volume at start of the simulation period (million m ³)
Line Type 32	Variable 1	Number of points for growth of inspoils storage where decant occurs
	Variable 2	Interpolation option
Line Type 33	Variable 1	Years for growth factors for inspoils (decant).
Line Type 34	Variable 1	Inspoils decant growth factors for specified years.
Line Type 35	Variable 1	Number of points for inspoils storage where seep through weathered zone occurs
	Variable 2	Interpolation option
Line Type 36	Variable 1	Years for inspoils storage where seepage occurs.
Line Type 37	Variable 1	Inspoils storage where seepage occurs for specified years.

Line Type 38	Variable 1	Maximum seepage rate (M ³ /month)
	Variable 2	Exponent of seepage equation
Line Type 39	Variable 1	Surface area of opencast pollution control dam at full storage (km ²)
	Variable 2	Capacity of pollution control dam for opencast pit (million m ³)
	Variable 3	Initial volume in the pollution control dam (million m ³)
	Variable 4	Initial concentration in the inspoils dam (mg/l)
Line Type 40	Variable 1	Standard deviation / Workings area
Line Type 41	Variable 1 to 10	Flow reference for Q vs. SLD relationship
Line Type 42	Variable 1 to 10	Load for Q vs. SLD relationship
Line Type 43	Variable 1	Standard deviation / seep and decant
Line Type 44	Variable 1 to 10	Flow reference for Q vs. SLD relationship
Line Type 45	Variable 1 to 10	Load for Q vs. SLD relationship
Line Type 46	Variable 1	Underground section name.
	Variable 2	Outflow route to central pollution control dam.
Line Type 47	Variable 1	Catchment area upstream of undermined area (km ²)
	Variable 2	Area undermined bord and pillar (km ²).
	Variable 3	Area undermined high extraction (km ²).
	Variable 4	Factor to calculate runoff from surface of high extraction mining
Line Type 48	Variable 1 to 12	Monthly portion of underground section that recharges underground water.
Line Type 49	Variable 1 to 12	Monthly bord and pillar recharge factors.
Line Type 50	Variable 1 to 12	Monthly high extraction recharge factors.
Line Type 51	Variable 1	Number of points for underground Mining Board and Pillar area growth data
	Variable 2	Interpolation option
Line Type 52	Variable 1	Years for bord and pillar growth factors.
Line Type 53	Variable 1	Bord and pillar growth factor for specified years
Line Type 54	Variable 1	Number of points for underground section: High Extraction area growth

	Variable 2	Interpolation option
Line Type 55	Variable 1	Years for high extraction growth factor.
Line Type 56	Variable 1	High extraction growth factor for specified years
Line Type 57	Variable 1	Load generation variables, Standard deviation / recharge
Line Type 58	Variable 1 to 10	Flow reference for Q vs. SLD relationship
Line Type 59	Variable 1 to 10	Load for Q vs. SLD relationship
Line Type 60	Variable 1	Discard slurry dump data
Line Type 61	Variable 1	Slurry dump area (km ²)
	Variable 2	Runoff factor for dump.
	Variable 3	Proportion of seep to dam and river
	Variable 4	Capacity of pollution control dam at full supply volume (Mm ³)
	Variable 5	Area of pollution control dam at full supply volume (km ²)
	Variable 6	Initial volume in pollution control dam (Mm ³)
	Variable 7	Not known
Line Type 62	Variable 1 to 12	monthly recharge factors
Line Type 63	Variable 1	Number of points for discard slurry dump area growth factors
	Variable 2	Interpolation option
Line Type 64	Variable 1	Years for slurry dump growth factor.
Line Type 65	Variable 1	Slurry dump growth factors for specified years.
Line Type 66	Variable 1	Load generation variables, Standard deviation / spill from pollution control dam
Line Type 67	Variable 1 to 10	Flow reference for Q vs. SLD relationship
Line Type 68	Variable 1 to 10	Load for Q vs. SLD relationship

12.6 Network file example

The file has the name TS.NET, which means that it is the network file for the TS network.

```
L.001 TS 1
L.002 C:\WR90\DATA\
L.003 C:\WR90\RUN\
L.004 N
L.006 Y
L.007      15
L.008 1RUY
L.008 4RRY
L.008 6RRY
L.008 5RVY
L.008 2CRY
L.008 7RUY
L.008 11RRY
L.008 10RRY
L.008 9RVY
L.008 8CRY
L.008 12RUY
L.008 15RRY
L.008 14RRY
L.008 13RVY
L.008 3CRY
L.009      3
L.0010     3 'Gauging Station 3 '
L.0011     'MRA2M50.OBS'
L.0010     14 ''
L.0011     'MRA2M49.OBS'
L.0010     20 ''
L.0011     'MRA2M45.OBS'
L.0012     1922 1987
L.0013     20
L.0014     1  1  5  0
L.0014     2  0  5  0
L.0014     3  5 15  0
L.0014     4  5  2  0
L.0014     5  1  4  0
L.0014     6  4  5  0
L.0014     7  4  3  0
L.0014     8  6 10  0
L.0014     9  6  9  0
L.0014    10  9 10  0
L.0014    11  9  8  0
L.0014    12  0 10  0
L.0014    13 10  7  0
L.0014    14 10 15  0
L.0014    15 11 15  0
L.0014    16 11 14  0
L.0014    17 14 13  0
L.0014    18 14 15  0
```

```

L.0014    19  15  12  0
L.0014    20  15   0  0
L.0015     5
L.0016    'RT' 1 'Q'
L.0016    'RT' 2 'Q'
L.0016    'RT' 3 'Q'
L.0016    'RT' 4 'Q'
L.0016    'RV' 5 'Q'

```

Line type 1	Variable 1	The network code. This code is a (one to three) character code to identify the network.
	Variable 2	The network file storage version number.
Line type 2	Variable 1	The input directory name. This directory must exist, and must contain the data files.
Line type 3	Variable 1	The output directory name. This directory must exist, and may be different from the input directory.
Line type 4	Variable 1	Indicator to show whether a debug file is required. A debug file shows intermediate results, and is mainly for program debugging during development. Enter N for no debug file, D for debug messages to the screen or Y to write debug messages to a file with suffix .DBG.

Line type 5 is obsolete – it is only required if one of the positive debugging options is used, i.e. if Line type 4 variable 1 is either D or Y.

Line type 5	Variable 1	Start period for which debugging is required, enter 0 to show initial values. Every month is a period, i.e. the second year's month 1 is period 13.
	Variable 2	End period for which debugging is required.
Line type 6	Variable 1	Indicator to show whether the user wants a summary file. Enter either Y or N. If Y is entered, a summary file with suffix .SUM will be created. (See line type 16).
Line type 7	Variable 1	Number of modules in the network.

One line type 8 for every module in the network, as defined in line type 7, in the order in which you want to solve the network.

NB. Note that a runoff submodel should be solved before the outflow of such a submodel is required by the more downstream module. The demands of an irrigation block should also be determined before its source module. As a rule of thumb, therefore, solve the runoff and irrigation modules first.

Line type 8	Variable 1	The module number, followed by the abbreviation for the submodel type, followed by Y to solve this module. If an N is given instead of the Y, the module will not be processed. In this way, a network may be shortened somewhat. However, if a module expects inflow generated by an upstream module which was not solved, the program will alert the user and stop. There are 4 submodel type
-------------	------------	---

abbreviations:

RU for catchment runoffs

RV for reservoirs

RR for irrigation blocks

CR for channel reaches.

Line type 9 Variable 1 The number of observation points. Any route can become an observation point if you have a file with observed flows for this route.

Repeat line type 10 and 11 for every observation point.

Line type 10 Variable 1 The route number of the observation point.

Line type 11 Variable 1 The name of the file that contains the observed flow data for the route described in line type 10. The simulated flows will be compared against the values in this file.

Line type 12 Variable 1 Year to start simulation.

 Variable 2 Year to end simulation.

 Maximum: 150 years of simulation.

Line type 13 Variable 1 The total number of routes in the network which is to be modelled.

Repeat line type 14 for every route specified in line type 12.

Line type 14 Variable 1 The route number.

 Variable 2 The source (or tail) module number as it appears in the list of linetype 8's..

 Variable 3 The sink (or head) module number as it appears in the list of line type 8's..

 Variable 4 The route cost. This is a dummy variable only, which may in the future be used to resolve deadlocks. Not used at this stage; enter 0.

Line type 15 Variable 1 The number of summary lines to be written to the summary file.

Repeat line type 16 for the number of summary lines defined in line type 15.

Line type 16 Variable 1 Either 'RT' for a route or 'RV' for a reservoir

 Variable 2 The route or reservoir number

 Variable 3 A "Q"

12.7 Rainfile example

12.7.1 Raingauge files

The following input file has the name 0438533.mp

```
L.001 0438 533 1923 1982 6567.  
L.002 0438533 1923 142 531 590 774 554 1000 238 178 0 0 132 383  
etc. up to 1982
```

There could be a gap repeated by lines similar to L.001 and L.002

Line type 1 Section number, Position number, start year , end year (all four digits) and MAP (in tenths of a mm)

Line type 2 Gauge number (combined Section and Position numbers), year (four digits), monthly rainfall figures (in tenths of a mm) starting from October and ending in September of following year.

12.7.2 Rainfile (expressed in terms of percentages of MAP)

The following output file has the name TEST.RAN .

```
B06 1930 1.79 17.66 14.99 17.82 14.56 14.09 17.56 0.00 0.00 3.33 0.00 0.00  
etc. up to 1994
```

I.001 Gauge code, year (four digits), monthly percentage of MAP.

etc. up to 1994

12.7.3 Report file

The following output file has the name TEST.REP .

```
AVERAGE RAINFALL ON CATCHMENT OF CODE  
B06
```

DETAILS OF RAINFALL STATIONS USED

SECTION	POSITION	MAP(mm)	PERIOD OF RECORD	LATITUDE	LONGITUDE
438	514	580	1930 TO 1977	26.34	27.48
439	203	660	1970 TO 1994	26.53	28.07

RAINFALL INPUT AS PERCENT M.A.P

YEAR	STNS.	OCT	NOV	DEC	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP
1930	1	1.79	17.66	14.99	17.82	14.56	14.09	17.56	0.00	0.00	3.33	0.00	0.00

etc. up to 1994

At the end of the file values are also given for the following :

AVERAGE
ADJUSTED
STD. DEV.

L.001 to L12 Headers as described above

L.013 Year (four digits), station number, monthly percentage of MAP and total percentage.

etc. up to 1994

13 TESTING OF WRSM/PITMAN

This section serves to describe the tests that were carried out on the enhanced WRSM/Pitman model, i.e. the model developed from January 2005 to March 2008. Refer also to the WRSM/Pitman User Manual, WRSM/Pitman Theory Manual and WRSM/Pitman Programmer's Code Manual (first 12 chapters).

13.1 Testing Procedure

Some code was developed by Dr Bill Pitman of the WRSM/Pitman team and some was provided by others. The following people provided code and did their own testing (on their own code/spreadsheet) prior to exchanging the logic to the WRSM/Pitman team :

- Prof Denis Hughes on the Hughes groundwater-surface water method ;
- Karim Sami on the Sami groundwater-surface water method ;
- Dr Chris Herold on the WQT methods of irrigation;
- Trevor Coleman and Pieter Van Rooyen on the mining module.

Logic developed by Bill Pitman who carried out his own tests included :

- Comprehensive wetlands algorithm;
- Streamflow reductions algorithm which makes use of Gush data and
- Daily time step methodology.

The former Stewart Scott team programmer JP Kakebeeke carried out his own extensive tests while developing the code. He completed the code in December 2005 and sadly passed away in February 2006. Dr Bill Pitman and Allan Bailey took over the task of debugging and correcting code that was not correct. Grant Nyland has been added to the team to assist with correction of bugs and the phase 2 changes (incorporation of databases, GIS Viewer, graph enhancement, incorporation into Spatsim, etc.) required to the model for the WR2005 and WR2012 projects.

Certain people offered to do testing on the WRSM/Pitman program as follows :

- Stephen Mallory on the WQT method of irrigation;
- Annette Wiethoff on the comparison between Hughes, Sami and Pitman groundwater – surface water interaction and
- Trevor Coleman on the mining module.

Dr Bill Pitman, Karim Sami and Allan Bailey spent a few full days ironing out a bug and ensuring that results were as expected. Allan Bailey followed up with Karim Sami on numerous occasions thereafter to clarify certain issues.

Similarly, Dr Bill Pitman, Prof Denis Hughes and Allan Bailey spent a full day to iron out a few bugs/changes and were satisfied following comparison of graphs of flows that the Hughes method was working correctly.

Most of the testing was carried out on the B41 catchment (Steelpoort River catchment in the Olifants Water management Area) which was also used for the WRSM/Pitman course.

Detailed testing was carried out against Karim Sami's spreadsheet. This involves taking certain data from WRSM/Pitman as input data to Karim's spreadsheet. In particular :

- Column C is the naturalised flow from WRSM/Pitman **with the Pitman model switched on** . Note that these flows will differ from those obtained if the Sami model is switched on.

- Column E is the Pitman S data which is obtained from saving these flows from WRSM/Pitman (File| Save| Module Time Series| Runoff Modules | Pitman S (Sami) in Save Time Series) again with the Pitman model switched on.
- Rainfall (column AC) and evaporation percentage of MAE (Column AD) data from WRSM/Pitman.
- Groundwater abstractions (if any) in Column AJ.
- Sami and other data (single cell yellow block)

Tests as described above were carried out and the WRSM/Pitman flows in the outflow routes of the Runoff modules were checked against the Final Runoff column BH in the Sami spreadsheet. They were accurate to within 1 decimal place. Testing details and information pertaining to WRSM/Pitman code are stored in SSI datafiles entitled "W00.JNB.000103 Project WRSM2000 CODING, Files 1 and 2"

Dr Bill Pitman and Allan Bailey also carried out ad-hoc testing and resolved a number of bugs and enhanced certain aspects of the model relating to defaults, documentation, etc. These are described below. There is a cross-reference in the code to "akbdate" i.e. akb18mar2008 for each change. Issues in blue have been resolved, those in black are as yet unresolved or have turned out to be of no concern.

WQT-SAPWAT method

This method was checked against the B31 tertiary catchment (C:\olifants\B31_new) for irrigation block 1. Refer to output files rain_debug, DF_debug and RRDIN_debug in the testing folder. Flows to routes 3 and 4 were also checked for the following :

- WQT;
- WQT with drought reduction factor;
- WQT-SAPWAT with drought reduction factor
- WQT with drought reduction factor

It was noted that some flows were higher with drought reduction factors switched on than in cases where drought reduction factors were switched off. This is possible when irrigation is fed from a reservoir.

As a further check, a channel reach (CR2) in B31 was checked with a large inflow so that the supply to RR2 was not restricted. In this case all the flows were either the same or less with drought reduction factors switched on than with drought reduction factors switched off.

WRSM/Pitman Daily time step

This version of the WRSM/Pitman model was created during the WR2012 project. Testing was carried out against the DOS version and the results were confirmed as accurate by Dr Bill Pitman.

WQT Type 4 Method

This method was introduced in 2015 based on enhancements made by Dr Chris Herold. The same changes were implemented in WRYM, WRPM and WQT. Full details are given in the theory manual. Four tests were carried out as given in the WRSM/Pitman Theory manual.

14 BUILDING THE WR2005 DATABASE AND CREATING AND RUNNING THE WR2005 INSTALLATION DVD (WR2005 SYSTEM ONLY)

There are ten WR2005 processes that can be run as follows :

- changes should be done on the S drive under Water/Project Data/Project Data Vision Projects ...000 to 999/JNB.000.000009 WATRES/Deliverable /Data. This updated data should then be copied to the C:\Wint\Projects\WRSM2000\WR2005\Data folder;
- the empty database should be copied from the S drive (... Deliverable/Source/Database) to the C drive (... Projects\WRSM2000/Database). Do not change the database name;
- build the program to modify the network datafile paths – refer to 14.1 ;
- run the WRSM2000 network datafiles paths – refer to 14.2 ;
- build the WRSM2000 databaseBuilder program – refer to 14.3 ;
- run the WRSM2000 databaseBuilder program – refer to 14.4 ;
- build the WR2005 Quat Results program – refer to 14.5 ;
- run the WR2005 Quat Results program – refer to 14.6 ;
- build WR2005Installation.exe datafile that is put onto the DVD for users to run to install the software and create the folders and datafiles required to run the dashboard – refer to 14.7 ;
- run the WR2005Installation.exe file – refer to 14.8 ;
- running the import rainfalls program – refer to 14.9;
- creating/editing the dashboard – refer to 14.10 and
- run the dashboard – refer to 14.11 .

These processes, where they are found and respective folders that are affected are shown in Figure 14.1 .

If data is changed, example rainfall, WRSM/Pitman datafiles, naturalised flows, etc. which should be done on the S (or F) drive, then the database should be re-built and the installation CD re-created. The C drive Data folder should be updated with the changes so that duplicate information exists.

14.1 Build WRSM2000 DotNetPaths Program

To build the EXE go into Delphi ver 7 i.e. Start/Programs/Codegear/ and File | Open Project| select c:\WINT\Projects\WRSM2000\WRSM2000 DotNetPaths. Then choose WRSM2000 DotNetPaths.dproj and choose Project | Build All Projects

14.2 Run WRSM2000 DotNetPaths Program

After the above, press the green button to run. Select c:\WINT\Projects\WRSM2000\WR2005\Data\WRSM2000 Network Model Data\WRSM2005NetworkList.txt . The program will then run through every network and change the path to the above.

14.3 Build THE WRSM2000 DatabaseBuilder Program

Only required if the structure of the database changes not the data itself. As for 14.1 but select the WRSM2000 databaseBuilder program.

14.4 Run the WRSM2000 DatabaseBuilder Program, i.e. Build the WR2005 Database

After the above, press the green button to run, select c:\WINT\Projects\WRSM2000\WR2005\Data\WRSM2000 Network Model Data\WRSM2005NetworkList.txt

. The program will then run through every network and set up the WRSM2000.mdb on c:\WINT\Projects\WRSM2000\Database.

The WR2005 Access database is set up from the normal WRSM/Pitman textfiles. The WR2005DatabaseBuilder application is used for this. If there are any errors they will be written to an error log file (see Logs folder). The Get Network from Database , Save Network into Database and Delete Network from database options in WRSM2000 | File| Database are used for other WRSM/Pitman networks that the user may set up. About 8 hours is required for this process but it will take longer if there are errors in the text files.

14.5 Build the WR2005 Quat results (quaternary catchment naturalised flows) program

Only required if the quaternary structure changes or if source code affecting the Spatsim database is changed. For example, various information is set up on the database path, etc. for Spatsim. This is controlled in the data dictionary 1. As for 14.1 but select the ImportQuatResults program.

14.6 Run the WR2005 Quat results program (Import quaternary catchment naturalised flows)

Only required if the naturalised flows change so that opting for this within SPATSIM will produce the correct flows on the map. After the above, press the green button to run. Choose the QuatNaturalFlowFiles.txt input datafile (on the WRSM2000 folder). This program modifies the catchments-sa.dbf datafile so it must be copied back to the F/S drive if this program is run.

If the format of the MTS files or database location or structure were to change (for example), go into Delphi and load the ImportQuatResults program, then load the relevant (“*.pas”) routine such as uResultFileAgent or uMapObjectsAgent, edit, save, build all and run.

14.7 Create the WR2005 Installation executable

Required if any information changes.

This is done using the software product “InstallAware” which is part of the Delphi version 7 (CodeGear). There must be an InstallAware folder under Program Files which is on the same level as WR2005 with its sub-folders. This enables the InstallAware code to access the information for WR2005. This WR2005 folder with sub-folders Data, etc. must be on the following directory: C:\WINT\Projects\WRSM2000. From c:\WINT\Projects\WRSM2000\InstallAware, run InstallAware.mpr. This datafile will also result in the WR2005 SPATSIM being installed. Then choose Projects|Build. This will create the datafile WR2005Installation.exe which will appear under the C:\WINT\Projects\WRSM2000\InstallAware\Release\Single folder which can be put onto a DVD ready for the user. About 2 hours is required for this process

14.8 Run WR2005 Installation DVD (WR2005 system only)

Run WR2005Installation.exe . The user will have this on a DVD. This program will check if WR2005 has been installed before and if so will spend about 15 minutes un-installing the old WR2005. Then the user will have to decide whether to opt for a complete or compact installation. There follows the installation to a folder which should be C:\Program Files\WR2005 which takes about another 15 minutes. This copies all the required information and sets up the dashboard.

14.9 Running the import rainfiles program

Not required at present as rainfall data is not included in the database. Provision has been made in case

this is needed at a later stage.

14.10 Creating the Dashboard

Only required if the structure of the dashboard changes.

The dashboard is a menu system for the WR2005 project to run models like WRSM/Pitman, view GIS maps, view reports, analyse spreadsheet data, etc. The dashboard is not a part of WRSM/Pitman but requires some knowledge of editing with Delphi (version Delphi 2007 used).

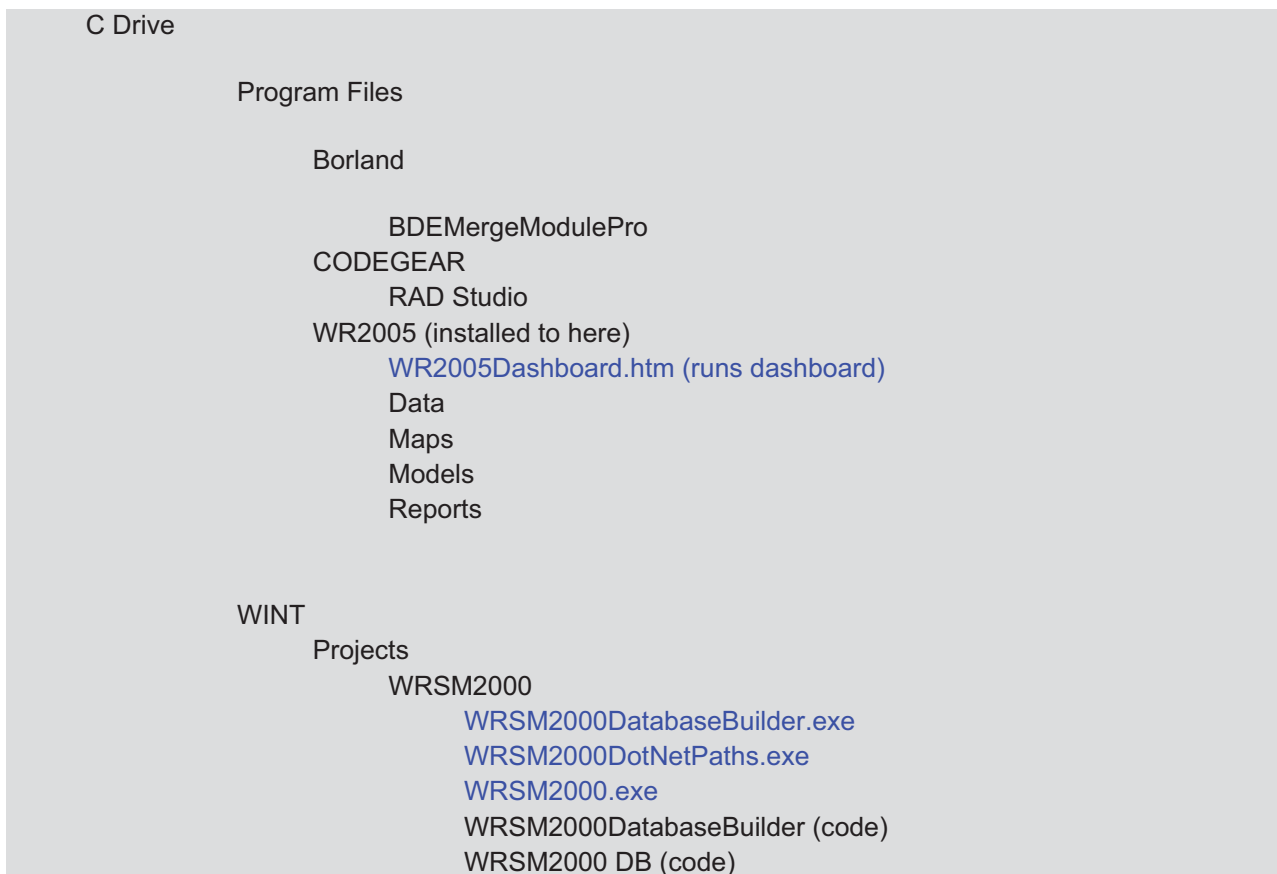
Using CodeGear (Delphi) from Program Files, go into File and open the datafile (html) WR2005Dashboard.htm from C:\WINT\Projects\WRSM2000\WR2005. There are three views, namely: code, design and history. The dashboard is edited via the code option. Then save and run. The Internet Explorer options on a particular laptop or PC might be set to block content and with this setting will not allow the + symbol to “explode” the GIS maps for example. To resolve this problem, go into Tools/Internet Options/Advanced/Security and tick the first 2 boxes (Allow active control for CD's and My Computer).

14.11 Running the WR2005 dashboard

To run, access the folder WR2005 off the C drive | Program Files and execute the WR2005Dashboard.htm datafile.

The DVD should also have ArcReader and Adobe Reader on it so that the user can run directly from the DVD if he/she so wishes.

Note: on several of the issues above : If any source code is changed in Fortran or Delphi, then run CompareIt and copy from C:\WINT\.... to the F (same as S) drive.



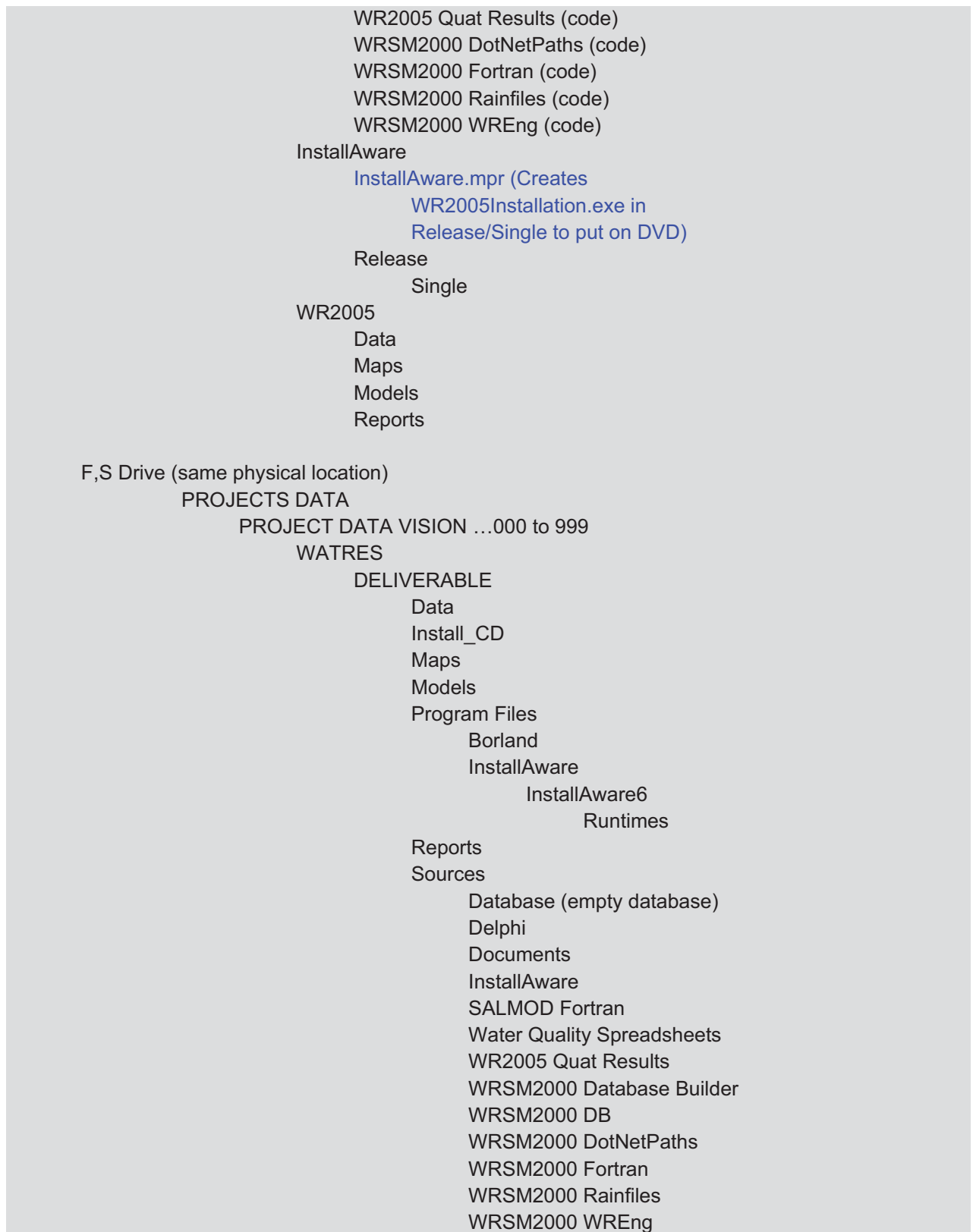


Figure 14-1: Diagrammatic representation of folders and datafiles

15 WEBSITE (WR2012 STUDY ONLY)

The website is a WR2012 innovation and was developed by Mr Tobias Goebel of Gtis. It replaces the DVD system used in WR2005. Mr Allan Bailey has administrator rights to register users and change or add to the website. The following are the main components of the website (www.waterresourceswr2012.co.za):

- home page which contains a register, login, resource centre, about WR2012, partnerships, contact details of Mr Allan Bailey, forum, website schematic and other buttons relating to the WR212 Launch as shown in Figure 15.1;
- website schematic which shows how the various menu items in the resource centre link together as shown in Figure 15.2;
- forum option for users to post comments and queries to Mr Allan Bailey for all to see and
- resource centre which contains a menu system of the following:
 1. GIS Maps
 2. WRSM/Pitman and Data Sets
 3. WR2012 Reports
 4. Quaternary data spreadsheets
 5. Patched Observed Streamflow Data
 6. Catchment Rainfall Groups
 7. Catchment based rainfall datafiles
 8. Rainfall stations
 9. Naturalised flow datafiles
 10. Water Quality
 11. Monitoring
 12. Land/ Water Use
 13. Present Day Flows
 14. Reservoir records/ Dam balances

The resource centre menu items are numbered to relate to the website schematic.

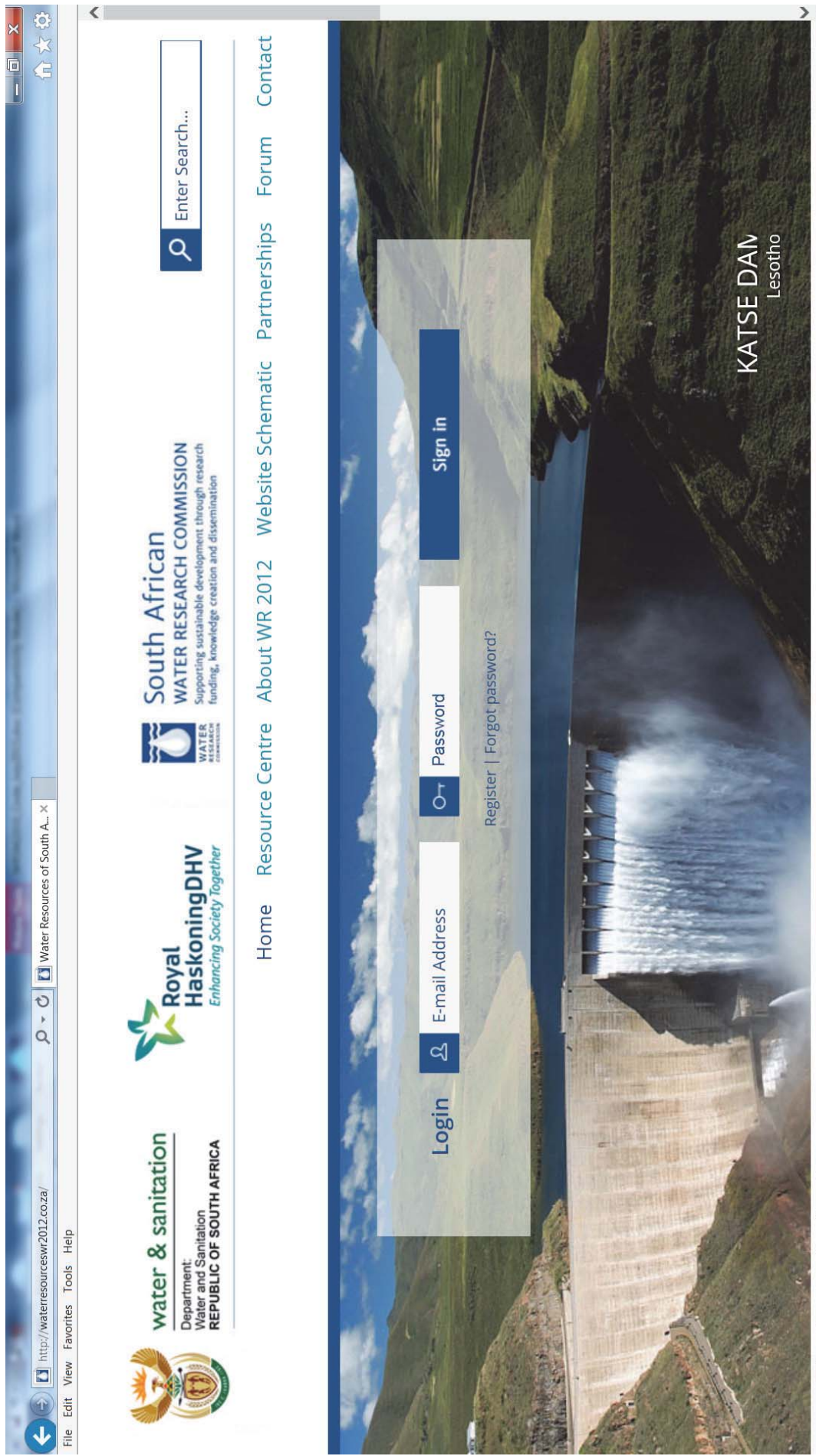


Figure 15-1: Website home page

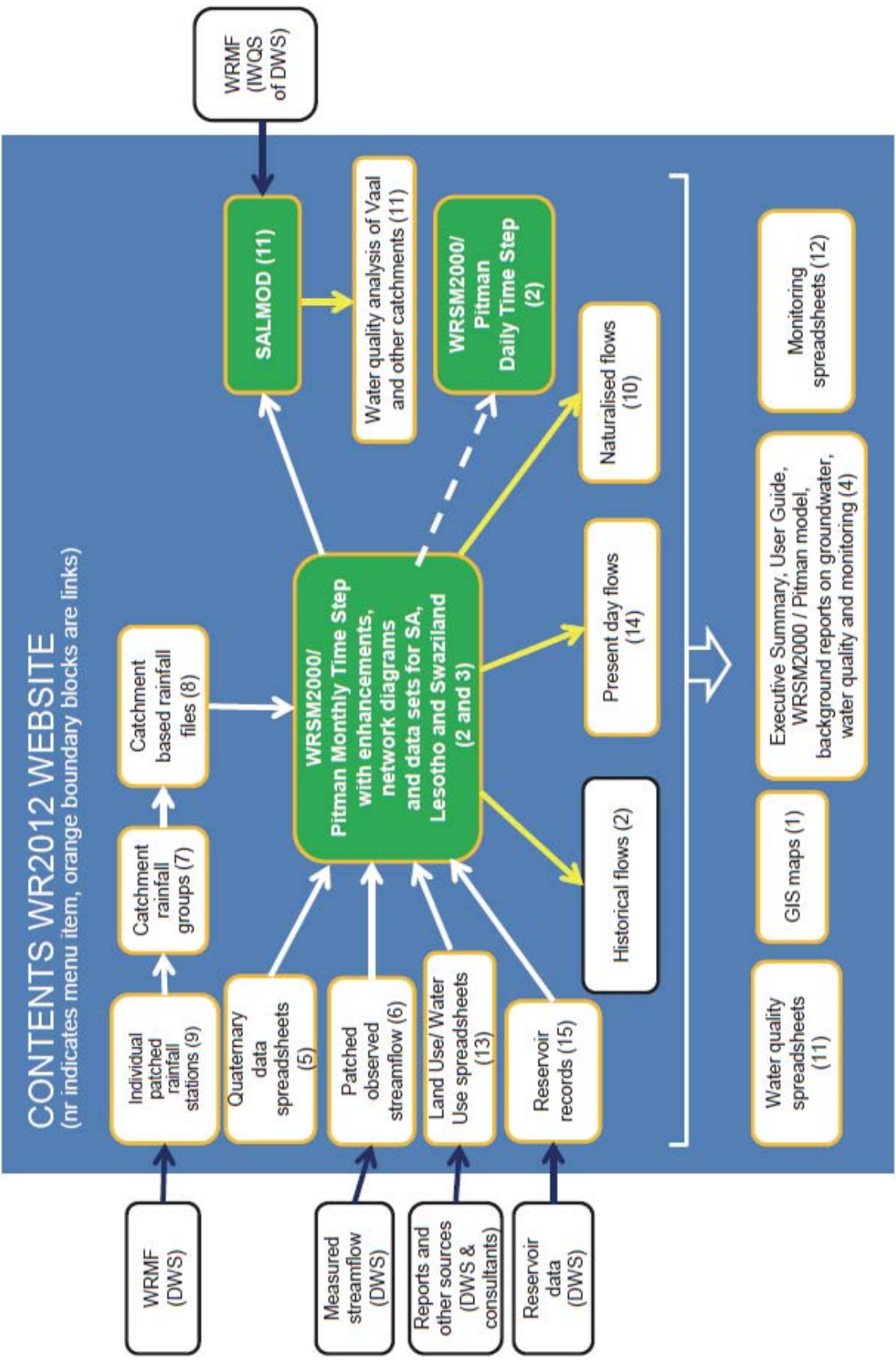


Figure 15-2: Website schematic showing links

As an administrator with rights, Mr Allan Bailey has access to change and add to the content. This is accessed via Admin on Mr Allan Bailey's login which gives the screen shown in Figure 15.3. To register a new user which comes as an e-mail request to Mr Allan Bailey, under "Users and Groups", the "Frontend User Management" option is selected. Then the "Users" tab is selected which shows those awaiting registration as "New Users". By using the "filter" option followed by "Apply", the registered members can be viewed.

To register a new user, the name is clicked on and a password is assigned, the "New Users" box must be unticked and the "Members" box must be ticked. Finally the "Submit" key is pressed and then the user is notified by e-mail that the logon is their e-mail address and what their password is.

Regarding changing the content, the "Content" key allows the administrator to change whatever needs to be changed or added. If datafiles have to be downloaded then the Filezilla software is used. The following procedure is followed after invoking Filezilla:

- Click on "Connect" icon in the top left corner to "open the site manager";
- Click on "public_html";
- Click on "uploads";
- Click on "resources";

This will produce the screen shown in Figure 15.4 below . The left-hand screen shows the folder system on Mr. Allan Bailey's server and the right-hand screen shows the website folder system. Datafiles can then be dragged from the left-hand screen to the appropriate right-hand screen folder. The website folder system has the following four folders:

- Data;
- Maps;
- Models; and
- Reports.

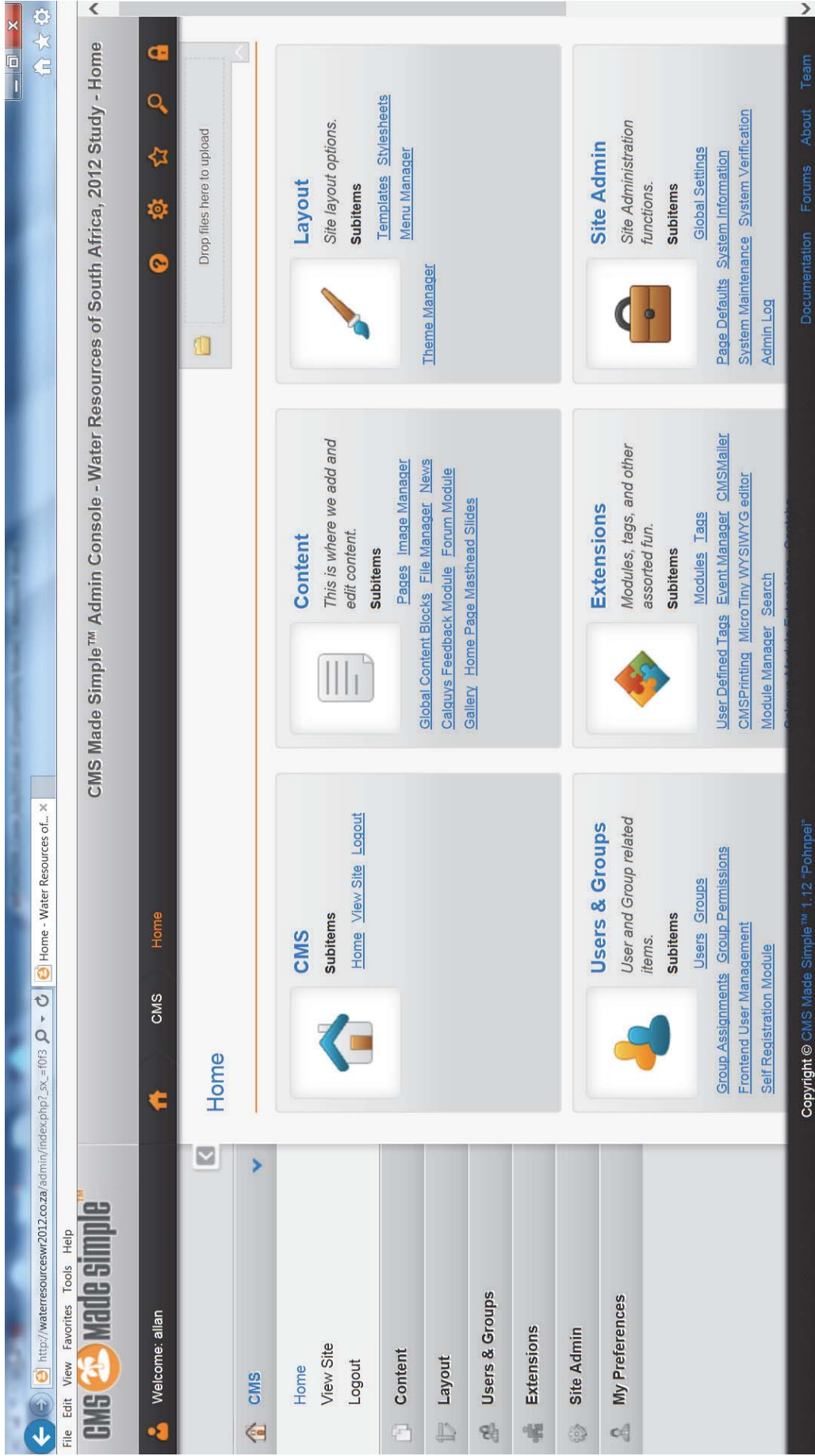


Figure 15-3: Administration view of the website

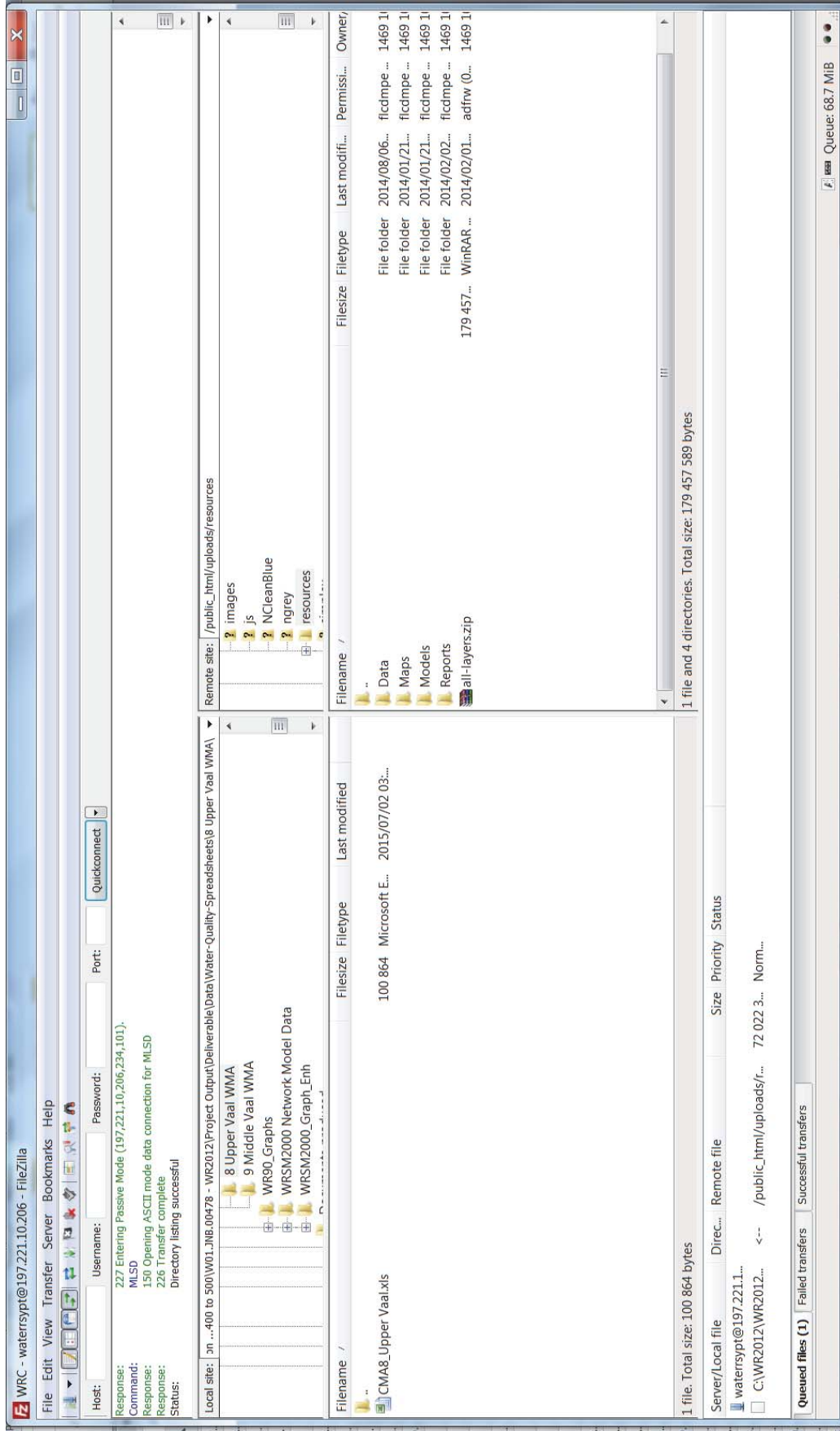


Figure 15-4: Filezilla screen to upload data

15.1 Adding GIS maps

GIS maps can either be downloaded singly or for the full set. The following procedure is required to update a map and make it available:

- Firstly the Coverages section must be updated which consists of shp, dbf, etc. files;
- Secondly the GIS_Maps section needs the mxd and pmf files. Note that a special licence is required to create the published map file (pmf).
- Thirdly a “start.html” must be set up which can be edited in notepad with the relevant map. See below for an example of a runoff map.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>WRC 2012</title>
</head>

<body>
<p><a href="GIS_Maps/WR2012 GIS maps/Figure_3_Runoff_2012.pmf">Click here to load the runoff
map</a></p>
</body>
</html>
```

- Set up Coverages, GIS_maps and the relevant “start.html” in a folder say on c:\runoff . See below

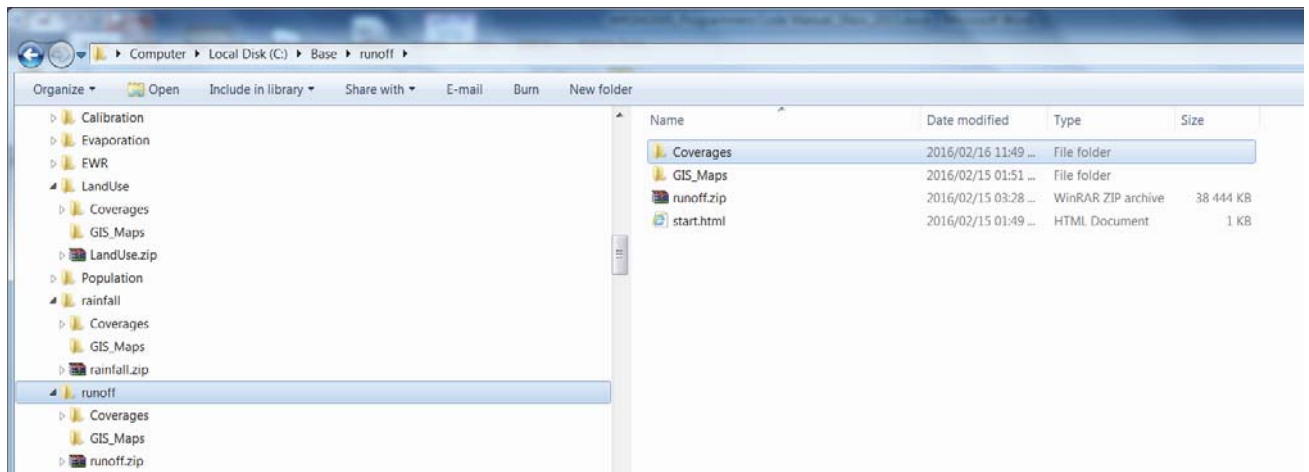


Figure 15-5: GIS maps folders

Note that the Coverages folder should contain all the sub-folders for that map as shown in an example for the runoff map below:

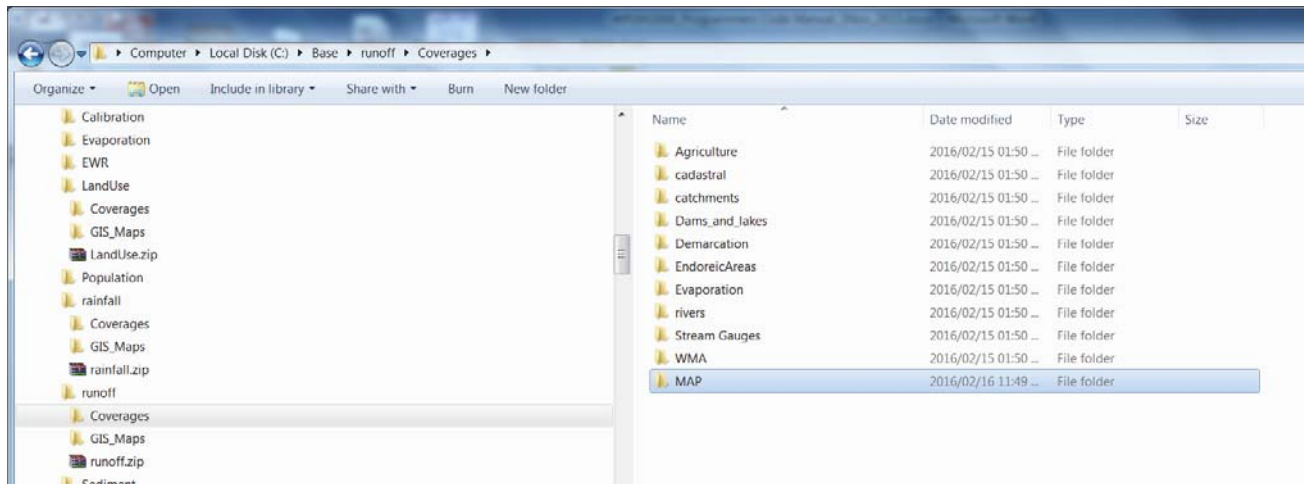


Figure 15-6: Coverages folder

Note that MAP is needed for runoff as it has the files for the shading of MAR.

- Now zip these 3 files using the EXE option (add to archive)
- Now go into Filezilla and connect, etc. to the Maps folder and then GIS_maps and then wr-2012-gis-maps
- Delete the previous runoff.zip file and then copy (drag) the new runoff.zip from c:\runoff in the left window to the right window (this will take about a minute or so). A successful transfer will be indicated.
- Now the link must be established in the website, under Admin rights, go to the Content, Pages, 1. GIS maps (WR2012) section, then go to the Content section by scrolling down and highlight (for example for runoff) the Runoff (39 Mb) part.
- Then click on the “Insert/Edit link” icon . This produces the following screen:

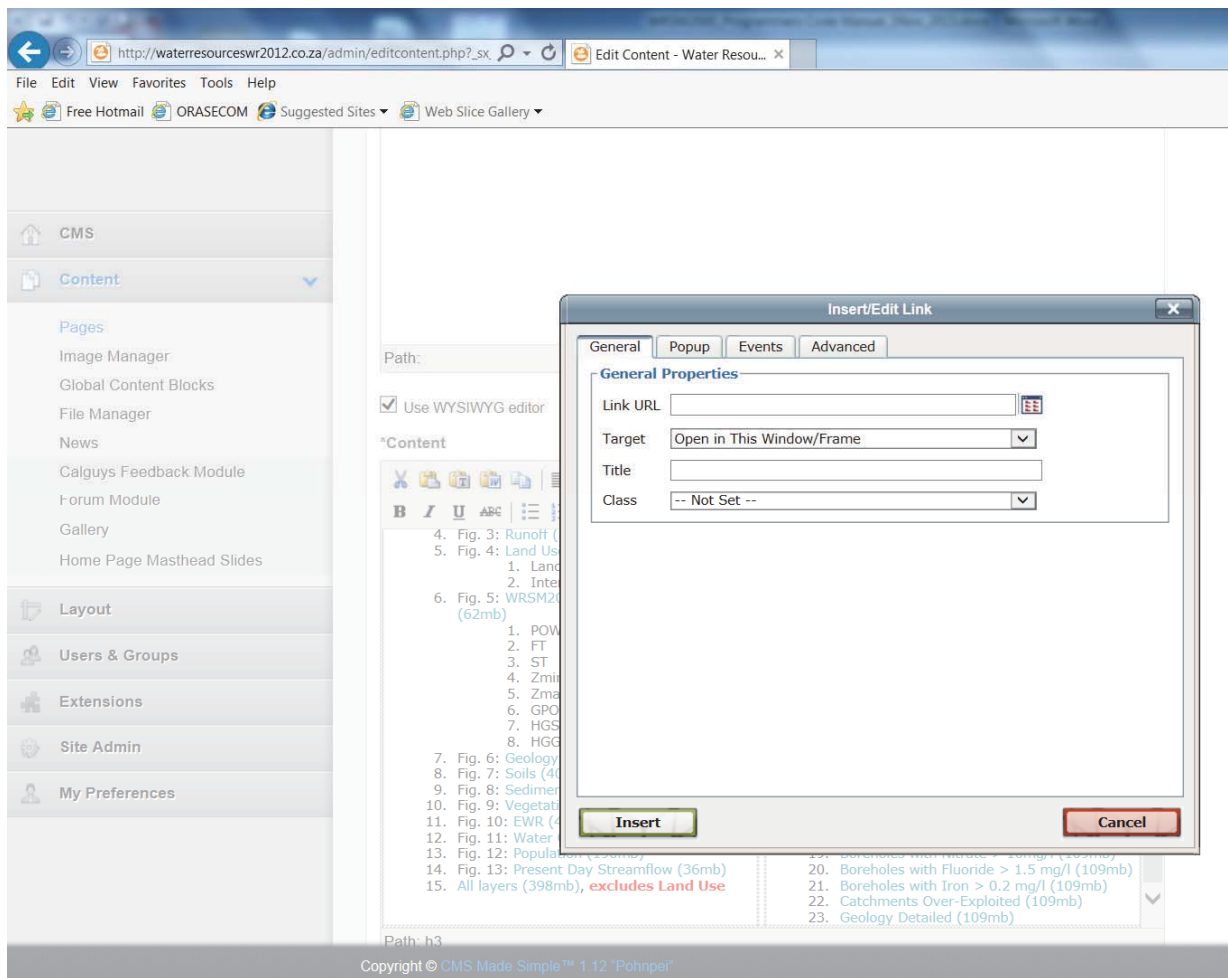


Figure 15-7: Website Insertion

- Now click on the icon for the Link URL and copy the path of the pmf file into it.
- Now click on the down arrow in Target and select "Open in New Window (_blank)
- Now press the Insert key, followed by the Apply key. Note that Apply and Submit are the same function except that Apply allows the user to stay in the current screen.
- Now logon to the website as a user and go to the GIS Maps section and download a map, i.e. choose either one map or all the maps and click on Save
- Exit the website and Filezilla and go to C:\downloads
- A runoff.zip file will have been saved there, extract the runoff folder to a suitable folder say GIS_test_runoff
- Now click on runoff to get Coverages, GIS_maps and "start.html"
- Click on start.html and "Click to load the runoff map" and then Open
- The map will appear

15.2 Google Analytics

A heat map is available to see which options are used the most from the website. The link is <https://insights.hotjar.com/h?site=49464&heatmap=224755&token=9f071e6aebccc828daf5a9b1c4e6f286&device=desktop&type=click>

However to get the latest update, a request must be sent to the web developer Mr Tobias Goegel.

Other interesting analytical tools can be obtained by:

- Login to Mr Allan Bailey's gmail account ([http://WR2012SA@gmail.com](mailto:WR2012SA@gmail.com))
- Then go to <https://www.google.com/analytics/> (at the top of the screen)
- Click on "SIGN IN" (see Figure 4)
- Then opt for Google Analytics or whatever you wish on the right hand side

Graphs and maps can be obtained to analyse the users use of the website

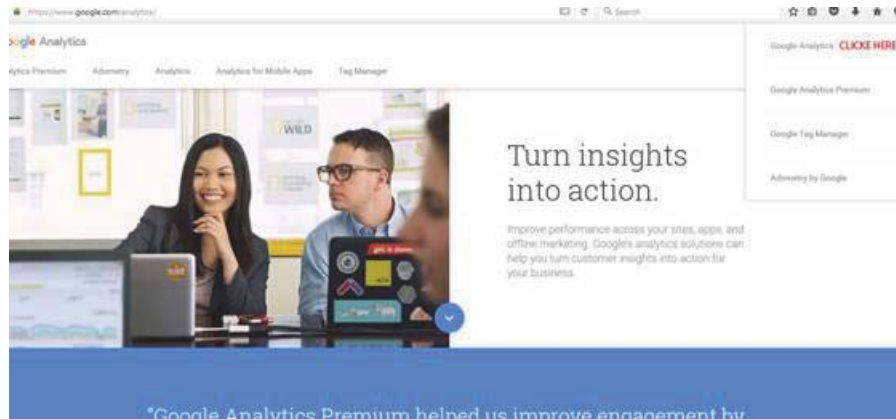


Figure 15-8: Google Analytics screen with further options on the right hand side

APPENDIX A

Enhancements/Bugs found/queries/suggestions

WRSM/Pitman model

Testing from January 2006

Bugs found/queries/suggestions (blue = resolved, black = possible pending issue for future reference :

1. Problems with the writing out of irrigation and that reported in the appendices i.e. format seems to have changed. Refer to E-mail sent to JP on 19/1/2006 09:08 datafile. Found by Allan – corrected code and programmer’s code manual. (akb10feb2006).
2. Problems with the writing out of irrigation to and subsequent reading of datafile error. Refer to E-mail sent to JP on 18/1/2006 17:00 , datafile. Found by Allan. Similar fix to point 1. (akb10feb2006).
3. Problem in code and on WRSM/Pitman screen with Irrigation module, effective rainfall limits 1 and 2 were the wrong way round. Corrected by Allan. (akb10feb2006).
4. Problems with Sami groundwater. I then tried playing around with unsaturated storage to see what happened. (As I mentioned in my previous e:mail, changing it from 0 to 10 had absolutely no effect.) I increased it from 10 to 50 (Karim's suggested upper limit for non-dolomitic areas) and, lo and behold, MAR shot up from 104 to 132 mcm and all flows were higher. I then tried unsaturated storage of 5 and obtained identical flows to that with it at 0 and at 10. I then tried 20 and got very similar flows to that for 50 – see above.

It appears from this that there is some magical value of unsaturated storage at which all flows suddenly increase significantly, i.e. in my case somewhere between 10 and 20. I have not bothered to find it as there is obviously something seriously wrong somewhere – wouldn't you agree?

Refer to E-mail sent to JP on 17/1/2006 14:31 , datafile. Found by Bill. Corrected after working session with Bill, Karim and Allan. It was an input data problem – Unsaturated storage should not be less than HGGW. Trapped in code with error message. (akb22feb2006). Also moved all “N column” code to after “P column” in two places and changed a statement regarding “N35 column” calculation. (akb16feb2006).

5. Croc West issue

Refer to E-mail sent to JP on 11/1/2006 15:58 , datafile. Found by Allan. A message to the effect of “Adjust return flow. Reduction factor cannot be negative, etc.”. Occurs with defined abstractions and irrigation blocks and not enough water. Corrected with additional wording applied to error message. (akb29mar2006)

6. For the Sami method in the calibration screen, HGSL and HGGW are not greyed out ?? Surely these are just Hughes parameters ? I left them as they were but when I applied, I got an error message about HGGW not being able to be zero. I also (prior to this error message) got an error stating that "FT should be greater than GW but for Sami we will allow it" – shouldn't we drop that error message if we are using Sami ?? Suggested by Allan. Refer to E-mail sent to JP on 9/1/2006 09:55.

Not sure if there still any problem here – Sami does appear to use HGSL and HGGW

7. In the Sami GW you have units of mm, as far as I can see it should be m (it is the same as the Hughes Rest Water Level). Found by Allan. Refer to E-mail sent to JP on 5/1/2006 11:27. All units and defaults were corrected by Allan after discussion with Bill. (akb20mar2006 and (akb22mar2006)).

9. For the Hughes groundwater, in some of the B41 runoff modules, the WR90 values for FT are zero. However, I left them = 0 and ran with defaults for all the other variables. When I checked simulated output I saw that MAR was very much higher than for Pitman model and that the baseflows were all very high when they should really be almost insignificant.

I then looked again at Dennis' document on calibration testing and saw that he had tested 2 catchments (E10K & Q92F) with FT = 0. He must have had reasonable results as there were no complaints about them. I also noticed that he had increased the default HGGW for these considerably – E10K from 1.2 to 10 and Q92F from 0.3 to 15!

I did not try such action in my case as, logically, this would surely increase baseflows, which are already too high. At this stage I'm not sure whether the problem is in our coding of Dennis' algorithms or not, but it does look as if that might be the problem. Found by Bill. Refer to E-mail sent to JP on 16/1/2006 13:46.

Bill added later that it seems OK if Zmin and Zmax are low values but if they are 999 (together with FT=0) then the bug occurs.

Akb5oct2006 Denis found two bugs. Awaiting Bill comment.

10. Interflow lag – Sami says it should not be there. Check that it makes no difference leaving it out. Cannot find E-mail reference to JP but it was sent.

It appears that it should be there.

11. Riparian strip evaporation (Hughes GW). In his e:mail of 17 Jan, Dennis said that – in catchments with FT=0 – groundwater contribution can be eliminated by making Riparian strip evaporation area large enough. In my test catchment I first used the default of 0.2%, which yielded a significant baseflow. So, I increased the evap area from 0.2% to 2% – a fairly hefty tenfold increase! However, the simulated flows came out identical to the previous run, which makes me think that the model is not losing any water from the riparian strip. It is something we must obviously check. Found by Bill Pitman. Refer to E-mail sent to JP Kakebeeke on 25/1/2006 14:34 (originally sent by Bill Pitman to Allan Bailey on 18/1/2006)

Denis Hughes has found no problems but is still to do further checks – not critical for release.

12. SFR iterations. In some runs with Hughes GW (on a catchment with afforestation) it did not converge after 40 iterations and gave me a message. (By the way, this took quite a while as it meant 80 runs for the Hughes method. When I looked at the stats of the slave they looked quite reasonable and close to that obtained by "Pitman GW". I don't know what closing error has been programmed into the code. I worked with 1%, which to me is good enough. Anyway, I have the following suggestion to avoid excessive iterations and error messages that might give the user grey hairs.

For iterations 1 to 10, work with closing error of 1% (on MAR & lowflow).

If we have not achieved errors of < 1% by iteration 10, then increase closing error to 2% for iterations 11 to 20.

If we have not achieved errors of < 2% by iteration 20, then increase closing error to 4% for iterations 21 to 30.

If after 30 iterations, we have not achieved success, then definitely tell the user about the problem. I also think that, after the iteration has been completed successfully, the number of iterations and actual vs target %reductions in MAR & lowflow should be reported somewhere.

- . Found by Bill. Refer to E-mail sent to JP on 25/1/2006 14:34 (originally sent by Bill to Allan on 18/1/2006). Corrected by Allan and Bill. (akb13feb2006 in wrsm2002.f90).
- 13. In the irrigation WQT method you cannot edit efficiency growth year and value. Found by Allan. Refer to E-mail sent to JP on 26/1/2006 13:14. Corrected by Allan. (akb10feb2006).
- 14. Alien veg. Bug in the riparian % (endless loop). E-mail to JP in drafts dated 6/2/2006. Subsequently discussed with Bill – it appears according to Bill that this is actually not a problem.
- 15. Deletion problem in Bennie's network. Lost entire network file except for a few lines. E-mail to JP in drafts dated 6/2/2006.
- 16. Sequence of modules in network. E-mail to JP in drafts dated 6/2/2006.
- 17. Query about sequence in network. Refer to E-mail sent to JP on 18/1/2006 17:12 datafile. Found by Allan.
- 18. WREng.dll compilation of correct version issue. Sorted out with Grant. Need to use Delphi to compile WREng.dll .
- 19. The SFR iteration procedure failed because it wanted to set PI to a negative value. I checked this on my spreadsheet as used for the tests on afforestation and got the same result. So, what I did was to build in a check for PI going less than 1.5 (and FF < 1.0) and to reset them to 1.5 and 1.0, then recalculate new value for SL and ST. Corrected by Bill and Allan. (akb13feb2006 in wrsm2002.f90)
- 20. Mining module – the module complains that a year is outside of the range. This is because the TERP2 interpolation routine is passed "nyear + 1" which is 1 year beyond the end year. Found by Allan and E-mailed to Bill and Pieter Van Rooyen 27 March 2006. Corrected by Allan by setting growth2 = growth1 for the year beyond the final year. (akb28mar2006).
- 21. As above with mining module – route 0 problem ? Only found by Allan. Solved akb26apr2006.
- 22. PI exceeding 15 for the course module. Found by Karen doing tutorial 1 and E-mailed to Bill 27th March 2006. Happens for alien veg and afforestation when an area of less than 0.4 is entered. Set up a message about this and set the variable = 0 in this case. (akb29mar2006).
- 23. Negative runoff due to alien veg ?? As above –linked to above problem. Found by Karen doing tutorial 1. Fix as above. (akb28mar2006)).
- 24. Checks for slave modules that catchment area is the same as one of the year-area pairs. Found by Allan. Trapped and show error message (akb29mar2006).
- 25. Space in filename ?? Reported by John 24th March. JP was meant to have fixed ?? Solved using Grant Nylands revised Eslash routine (akb20apr2006).
- 26. Does not read from the server – missing a slash after the last sub-directory. Solved using Grant Nylands revised Eslash routine (akb20apr2006).
- 27. Does not show tips in View Statistics in Sami or Hughes. Bill aware of this – this is to be updated once we have sufficient knowledge.
- 27. SSI logo on screen. Removed for now as it would not compile with this in (akb2006). Grant developed Delphi dll to link in but this did not work (possibility that Lahey old version does not

- support Delphi dll's). Used JP coding and new SSI/DHV bitmap to change resource.rc, resource.F90 and WRSM2001.F90. Grant changed Settings : Compiler/Linker Options but re-set. Comments only in WRSM2001.F90 (akb4May2006). Not possible to add a picture.
28. Print Mining modules. Said Reservoir in the window. akb31mar2006.
 29. Mining module. If you do not have info on underground section, it can tell you that a route cannot be processed. Karen found this. Improve error message ?? Can't duplicate
 30. When adding a slave for alien veg. Complained about master not having split for tall, trees, etc. ?? Can't duplicate
 31. A whole lot of error strings had been set up in wrsm2004.f90 but not set up in wreng.dll i.e. getstring(2148) up to getstring(2175). I set up error messages in the code. This showed a warning if FT=0 for a master catchment instead of preventing FT from being zero. Corrected akb4apr2006 (error strings) and akb5apr2006 ("hard error" trap for FT=0 for master).
 32. If one puts 100 % in afforestation for eucalpts (second row) it puts strange values all over the place including in alien veg. Does not happen for pines. Solved akb6apr2006.
 33. If one changes calibration parameters, save the network, and go back into the calibration screen it has disappeared. If one closes the network and go back in then it is there. Found by Karen. Solved akb10apr2006. Details added in the code manual.
 34. E-mail from Bill dated 19/4/2006 regarding conversion from A-pan evap to A-pan factors for iirig. JP did implement coding but not sure if it is being used correctly/appropriately. Check with Bill when he is here ?
 35. Mine module. Washington found a problem if you have opencast and underground, it appears that each underground section must have a separate outflow route and you get a major problem with routes if you make them the same. It appears that the underground section should check that its outflow route should not be the same as for the mine as a whole. Solved 28apr2006. Added an error check.
 36. Mine module. There appears to be a problem if a second mine with one underground section is added and the program complains about section 2 not existing ?? IF you escape, the program does however run ?? Refer to B52 and XX networks. Solved 28apr2006. The wrong parameter was being passed to MM_UGCheck and the program was reporting an error (no underground section 2) on a second mine in the same network when there was no error. The same applies to opencast and slurry dump sections.
 37. Mine module. Allan found that if a slurry dump has no area then it complained about an underground mine (high extraction area) and would not allow you to correct. Found that the error had the wrong wording but it should allow you to correct area. The error message also does not mention that the area is zero – need to add to the error message "or the area is zero". Akb8jun2006
 38. Channel module for diversions – incorrect wording reported by Bill. Resolved on 2/5/2006 by changing the resource.rc file.
 39. Bill has concerns about Pines and low flow reductions issue. His E-mail of 2 May 2006 has been sent to Mark Gush for comment. Mark Gush did comment – subsequently decided not a problem (check).
 40. Naturalisation issue. Only for runoff sub-models. Is Slave working properly. Karen E-mailed diagram with flows to Bill to consider.

41. John reported some intermittent error with gauging stations disappearing. John to send an E-mail if problem is confirmed.
42. Problem with irrigation module, interpolation option does not exist. This resulted from an interpolation option of 2 (exponential) which the program set to 3 because there is internal manipulation between WQT and WRSM/Pitman and the KMA variable (interpolation option for allocation) was being read and written to file twice in Read3 and Write3 functions. Solved akb5may2006.
43. Problem in May course of GW and GL being used by Sami and changing statistics. This is because Sami requires a run of the Pitman model first (RU_GW function). Set GW and GL to 0 before calling this function in a Sami run. Solved akb11may2006. Also made GW and GL inactive for Sami method . akb12may2006.
44. Re-set colours based on the defaults in section 9 of the user manual. Introduced pink as well for category 1, blue was changed to be category 2 with white being category 3. akb12may2006. Changed both wrsm2004.F90 and resource.rc (for those that were permanently required such as Sami and Hughes GW screens).

+++++

Files in both version 2 and 7 should have all the changes up to above the same.

Differences from about end June 2006 are TiGenLib (Aton routine (akb13july2006)), WRSM2001 (akb5july – fresult lines), WRSM2013(split up and add back).

Need to copy from v2 to v7 the following :

- a. WRSM2007
 - b. Resource.rc and resource.f90
 - c. WRSM2000
 - d. WRSM2016
45. Chris Herold's comments entered in course manual. In particular as follows :
 - a. add a crops column to Crops2 dialogue. Done akb17july2006 but need to manually add crop types at the end of the line to data files that were est up with WQT method using version handed out at the course. WRSM2000, WRSM2007, resource.rc and resource.f90 changed.
 - b. Model connectivity check – just show ones that are invalid.
 - c. For slave modules , if afforestation then grey out alien veg and vice versa. Allan also noted in B60 (Wandile) that he had entered alien veg and afforestation in some slaves – it complains about areas of afforestation not being the same (even if you have zero) until you delete the years altogether.
 46. Changing/adding reservoir data. Complains about too few data points ?? when it has enough. Accepts input though – just get rid of irritating, incorrect message. From Alice and also seen by Allan.
 47. Widen some windows so that the filename can be read. Not possible to widen enough.
 48. Bill obtained an error message (twice) when running KP's T11 network is as follows:

"The initial aquifer storage can not be greater than the aquifer capacity of 10.00 mm (warning only, you may proceed)" It doesn't say which module, so I looked at all the runoff modules. They all had an unsaturated storage capacity of 20.00 and an initial unsaturated storage of 10.00 mm, so I wonder how this message popped up? Solution is to lower initial aquifer storage less than aquifer thickness * storativity.

48. Rainfall creation : if path too long it chooses the wrong station for the right hand window. Extend path length. Increased MNFILC parameter in WRSM2000 from 200 to 300. Appears to work now. akb12july2006. Wrsm2016 changed.
49. Mine module. Various areas should be able to be set to 0. Also bug in in spoil storage flowing out in the first month and area of mine not taken off runoff sub-model area. Identified by Trevor Coleman and Manie Mare – turned out to be a data error. Resolved areas not being able to be set to 0 on akb12july2006.
50. Default for storativity should be 0.1 not 0.02 akb26july2006.
51. Version 7 change only : akb31july2006. Added a deallocate statement in wrsm2004 to stop an "out of free contiguous memory" error when editing a runoff sub-model twice.
52. Irrigation : WQT method. IF you have zero for a percentage in Crops2 screen the program should warn the user that it does not add up to 100.
53. Increased alien veg limit from 15 to 20. akb4august2006.
54. Other Error message after 40 iterations to be checked by Bill. Bill's change to wrsm2002 for lack of convergence of iterations akb14aug2006.
55. Washington reported an error with ATOK mine in that "Invalid underground section number 2 in MM_GetSubArea . High Extraction Area " appeared as an error message. Cannot duplicate at a later stage. ??
56. Default for percentage (sugar cane) in Crops2 screen is 1.0 ? Should be 100.0 i.e. Bill confirmed it is a percentage and not a factor. Akb15aug2006.
57. Added groundwater graph akb17aug2006, akb18aug2006 and akb22aug2006.
Changes to title akb6oct2006
58. Changed all references to Master and Slave to Parent and Child. akb28aug2006 in resource.rc, wrsm2002.f90 and wrsm2004.f90.
59. Increased FF check from 1.5 to 1.55 (from Bill Pitman) akb31aug2006.
60. Added groundwater baseflow (discharge) and interflow curves to groundwater plot. Akb1sep2006.
61. Changed 2 equations according to Karim Sami . akb14sep2006.
62. Trying to add a route that previously existed results in a route 0 being created that is very difficult to correct. It should not create a route 0. Solved akb27sep2006 in WRSM2002.f90.
63. Tried to add logo to graphs in WRSM2014 but not successful. Akb27sep2006.
64. Changed to SSI new logo 27september. Just changed ssi.bmp file and re-compiled.
65. Changed Hughes method according to Denis' changes akb10oct2006.

Also changed Hughes method according to Bill's changes akb11oct2006.

66. Changed descriptions of groundwater plots and some comments in the code accordingly akb12oct2006.
67. Corrected some graph anomalies with spurious lines and thicknesses akb19oct2006. Also description of plots.
68. Fixed a spelling mistake in resource.rc (slurry dump). Only resource.rc changed. 30/10/2006.
69. Changed to November version message akb31oct2006. Second version since the passing of JPK
70. WQT-SAPWAT method. Added akb3nov2006. see also wvp2nov2006 and wvp6nov2006. Changed WRSM2007.f90 and resource.f90.
71. Drought reduction factors. akb7nov2006 and wvp6nov2006. This affected WRSM2000, WRSM2007, WRSM2015 and resource.f90.
72. Changed hydrological to hydraulic in the Sami screen on 10/11/2006.
73. akb14nov2006. Completed WQT_SAPWAT method including a change from Bill.
74. akb22nov2006. Changed error message on PI > 20.0 to a warning.
75. akb14dec2006. Changed one line in the WQT-SAPWAT method. ET should not exceed 75 mm.
76. Add WRC and DWAF logo's to main screen. Added to ssi.bmp on 2/2/2007.
77. In printing of plots, if one chooses dashed lines it applies to both observed and simulated. It should be solid for one and dashed for the other. Gn12feb2007 to wrsm2014.
78. Manie Mare reported a groundwater problem when there are groundwater abstractions. Corrected akb14feb2007. Had to correct code for column BA to use BG and not BF and also had to move code for column BA and BD and AR moved to beyond code for BG. Correct comparisons were determined using c:\ManieMare_problem2007\Data\b20aAbs_akb.xls column BH Final Runoff against the datafile SC1RU1_ncr.MTS (net catchment runoffs printed from WRSM2000 for runoff module 1). Aquifer storages have no zeroes unlike before which incorrectly showed negatives. Aquifer storage minimum (33.61 from SC1RU1.aqs agreed with Column BD in the spreadsheet 33.57).
79. Also following on from above but not linked to above is commenting out code for H35 where H35 = D35_interflow .. akb14feb2006 as well.
80. Anne Beater reported a problem with User Defined and other methods – see Simulation Error File. Caused by more than 25% of zeroes in afforestation area data. Can correct by setting afforestation area = 0.5 but Bill to set up a more elegant fix. Gn27feb2007 – Added GETNUP, etc. to get area, wrsm2004, wrsm2006 and wrsm2013. akb27feb2007 added comments. Gives some errors in SFR iteration in some catchments (B60) – Bill investigating. Resolved akb7mar2006.
81. Corrected Pitman S calcs for TS_GroundwaterStorage using SMEAN parameter akb12feb2007. Now can write out Pitman S values.
82. Child afforestation (and possibly alien veg) cannot start with its maximum value in 1920. This produces PI error messages and route flows screaming off to infinity. Correct with forcing users to start in 1920 with 0. akb19fen2007. Only afforestation changed, alien veg does not appear to give flows that have stars (very high values).

83. Static water level and aquifer storage cannot be identical as it causes a divide by zero error. Force users to have the aquifer capacity greater than the static water level (aquifer capacity is the product of storativity and the aquifer thickness). Akb19feb2007.
84. WQT-SAPWAT method. Add constraint condition from Bennie Haasbroek. Akb16feb2007
85. Add No to all button for MAP check. Gn12feb2007 to wrsm2000, wrsm2001 and wrsm2010. Resolved with synchronisation No. 3. gn8mar2007 as well.
86. Remove UNOFFICIAL COMPILATION MESSAGE. This related to the TiSD.lib file. Very complicated – Grant still possibly to look at in more detail.
87. Changed check for SL and ST so that the two cannot be equal. Akb19feb2007.

Integrated No 1 session. Grant Nylands code with his initials with Allan Bailey's code 21/2/2007. Grants additions were left as gn ..followed by date. Some code could not be commented on in modules resource.rc, lglib WRSM2000.WPJ and resource.f90. If necessary comparison can be made with Windiff on version of 20feb2007 in WINT_20feb2007 under Olifants/Task 8, etc. Also refer notes in WRSM2000 file dated 21st Feb2007.

This version bombed out on some catchments – Grant Nyland to resolve.

88. Added database features gn1jan2007 to wrsm2001.
89. Added network diagram gn12jan2001 to wrsm2001. Show a network diagram. Grant to fix path – fixed gn27feb2007.
90. Manie Mare requested that aquifer storage and static water level can go to the 3000 level. Need to change from 6.2 format to 7.2. akb23feb2007. Changed aquifer storage, static water level and unsaturated storage to F7.2.
91. Manie also reported another groundwater error in E-mail on 21/2/2007 regarding baseflows not being zero when aquifer storages were less than the static water level. Investigated together with Karim Sami and found that there was no problem. Manie should have been using 100m from river to boreholes and not 1000 m (in order to get baseflows = 0 a lot of the time. I.E. Baseflow can be greater than zero if aquifer storages are less than the static water level if the distance is large e.g. about 1 000 m.
92. Enable writing of data to SPATSIM gn18jan2007 to WRSM2001.
93. Changed compile and linking environment to match Grant's – done more by mistake in order to solve problem when clicking on NoToAll button which was actually the problem. Done on 23rd February 2007.
94. Comprehensive wetlands model – will not write to more than 1 channel. Problem with writing debug info to File. Akb23feb2007 in WRSM2003.
95. Bennie Haasbroek requested that error message when entering a crop factor of 0 be removed (made it a warning only). Akb23feb2007 in WRSM2007.
96. Added runoff module to some checks 23feb2007.
97. Manie reported an error in the mining routine if the mine area was very close to the catchments area. Awaiting more detail and test by Trevor Coleman.

98. Corrected spelling in transpiration. Resource.rc file
Integrated No 2 session. Still have Access Violation error.
99. Integration No. 3 session. GN has fixed Access Violation error GN8mar2007 (wrsm2001 and wrsm2010) and also moved folders and files. Also included Delphi code in Delphi folder.
100. Corrected viewing network diagram due to access violation error akb19mar2007.
101. Bill added some coding on SL. Akb10apr2007 and Wvp06april2007.
102. Commented out forcing users to enter afforestation for 1920 as zero as this appears to be a Sami problem. Akb10apr2007.
103. Normal method of irrigation – does not seem to retain areas. Possibly a hiccup with new WQT-SAPWAT method being added. Fixed akb25apr2007 but Cancel button retains areas with or without this change ?
104. Estelle Van Niekerk reported problem with paved areas growing. Corrected by adding new variable to wrsm2004.f90. See akb24apr2007.
105. Grant Nyland updated resource.rc, resource.f90 and wrsm2001.90 on 25apr2007. Look for gn25apr2007. One change was a status bar update and the other a version number introduction. In this respect version.rc was added. A SPATSIM layer number was included (LFeatureCode = 8).
106. Bill Pitman gave revision to WRSM2002. akb30apr2007.
107. Added warning about alien veg of less than 10 years of age producing iteration problems. Akb2may2007.
108. Some changes to tracefiles and trace statements seemed to have caused the model to slow down by about 10 times. Solved on 2 May 2007 by commenting out a few things and changing TRACE = .TRUE. to trace = .FALSE.
109. Bennie Haasbroek query about net and total return flows. Found runoff area not being adjusted with total return flows. Corrected 14may2007 . WQT-SAPWAT method had been excluded in wrsm2002.f90.
110. Manie Mare mine problem with negative flows. Traced to the growth2 variable that was going negative in the final year. Corrected growth2 calculation akb21may2007. Also increased format field of OCMMADAW and MMVSDS.

For mining areas to be deducted from the catchment area, refer to the User Manual. These areas were checked for both master and non-master runoff modules. Refer to c:/ManieMare_mine_problem_may2007/data/stk.net system.
111. Ninham Shand problem with stars alternating with very small values revisited. Warning added . akb22may2007 for static water level/capacity values exceeding 90%. Advise to use Sami revised SWL's in User Manual.
112. Synchronised with Grant (he added help options to pull in the user Guide, etc.). No need to update WRSM2000 at this stage. Affected resource.rc and WRSM2001.f90. 27 July 2007.
113. If an irrigation block is attached to a child runoff module, it complains about a sequencing error and you have to use Task Manager to get out of the model. Add an error message that child modules cannot be attached to irrigation modules (in the groundwater property tab). Corrected 29aug2007.

Not required in Sami , SFR children routine as JP has already put an error trap in there. 21sep2007 Changed HowManyParent references/functions to HowManyNonChildren.

114. Added an extra character to the network filename so that KP work can be saved to different network filenames and not all V11 for example. Akb21sep2007. This affected WRSM2012.f90, wrsm2002.f90 and resource.rc
115. Added an enhancement to the irrigation allocation screen for WRP: To allow all months to be reduced or just peak months to be clipped. Akb5oct2007 (for resource.rc) and akb16oct2007 (for wrsm2007.f90). Affected resource.rc and wrsm2007.f90.
116. Refined error trap for WQT_SAPWAT . akb2nov2007
117. akb13nov2007. Corrected error trap for only one year of allocation in the irrigation module.
118. Corrected the columns in Irrigation /Allocation year/value grid shifting to the left and the year being unseen. Changed resource.rc by clicking on year/value grid (in edit dialog mode), choosing General and changing the width of columns from 122 to 125. Date 27/11/2007.
119. akb24jan2008. and akb1feb2008 Changed “Number of months to average recharge” from f7.2 to i4. Also changed SL to HGSL and GW to HGGW and extended string lengths.
120. Add afforestation method to print of runoff module. Currently also does not recognize that a module is a parent and that it has a slave which is either alien vegetation or afforestation.
121. Prevent the user from putting in nonsense for POW and GPOW i.e. 0.
 - a) If FT = 0 then POW should be 3 (0 is not valid). This, however, makes no difference to the results.
 - b) If FT > 0 then POW should not be zero and this does affect the results.
 - c) If HGGW > 0 then GPOW should not be zero and this does affect the results.
122. Grant Nyland made some changes to the version number in Help|About (gn3sep2008) Also on 22/September, telephone numbers were updated and the SSI logo was updated and changed from an icon to a bitmap. This affected version.rc, resource.rc, resource.f90, wrsm2000.f90 and wrsm2001.f90 and a file was added SSIsmler.bmp.
123. akb14apr2009 and akb15apr2009. Corrected the WRSM/Pitman method of irrigation in 3 places with rdperc=1 instead of rdperc=0 in WRSM2003.f90 and wrsm2005.f90.
124. akb7may2009. Added code for Software licence code. Also changes/additions to Delphi folder and WRSM2000DB.dpr and WRSM2000.WPJ file as the Linker option in Settings had to have VerifyLicense added. WRSM2000DB folder datafiles were changed as well.
125. Added a Save All button so that all '.ANS' files could be saved, akb27sep2010 to WRSM2001.f90.
126. akb 27sep2010 Multiple ANS file save. Used wrsm2001.f90, wrsm2006.f90 and resource.rc. Ver 2.6 (Daily WRSM2000 is ver 2.5, version prior to multiple ANS file save is 2.4).
127. akb31may2011. Corrected wrsm2014.f90 so that LATITUDE appears on the Station graph and that Net catchment runoff does not appear in the legend.
128. akb11august2011 Ignore groundwater abstractions if the naturalised tickbox is checked for a runoff module.

129. akb18jul2012. Added a groundwater time series abstraction option. There is now a box where none, annual or time series can be selected from. See special note 10.21.
- 129 akb25sep2012. Corrected bug found by Caryn Seago relating to the Sami parameter groundwater evaporation factor not being correctly re-calculated if the catchment area is changed live. Set up a new version 2.7.
- 130 Brought in Royal Haskoning DHV logo instead of SSI. Changed 4 October 2012 but no comment line. In resource.rc go into the text editor and remove the SSI bitmap by replacing with this line :
- ```
SPLASHLOGO BITMAP DISCARDABLE "RHDHV_SplashLogo.bmp"
```
- 131 Brought in new graphs for rainfall from Daily version to new monthly version (akb11sep2014 in wrsm2015.f90).
- 132 Brought in new graphs for Plot menu to new monthly version (akb14may2012, etc. in wrsm2014.f90 and resource.f90).
- 133 Brought in calibration of runoff module enhancements to new monthly version (gn26jun2012, akb3jul2012 to wrsm2004.f90 and resource.f90).
- 134 Changed months to average recharge limit from 36 to 240 to new monthly version.
- 135 Brought in Type 4 WQT Irrigation method (akb23jun2014 to wrsm2007.f90, resource.f90).
- 136 Brought into **Daily version only** the daily time step method (mainly wrsm2004.f90, numerous dates, wrsm2014.f90 akb24jan2013, etc., resource.f90, etc.).
- 137 Brought into **Daily version only** the conversion from SAWS and DWS formats for daily rain and daily observed streamflow (akb27sep2014 in wrsm2002.f90, resource.f90). Tested with W2H032\_d.txt on c:\Daily\_B42\_Landuse.
138. In July 2015 Mr Grant Nyland substantially rewrote wrsm2014 and wrsm2015 and made major changes to wrsm2000 and wrsm20001 for enhancement of graphs with no line comment where changes were made.
139. On 15 September 2015, changes were made to the WQT Type 4 as discussed with Chris Herold, Bill Pitman, Caryn Seago and Manie Mare. These included screen description changes, addition of units, correction to the Climate grid for maximum rainfall factors, correct of the RR\_Adj\_Ret function (called from reservoir and channel modules incorrectly), greying out of some parameters not needed in Type 4), correction of the code causing the Apan factor to revert back to what it was before after pressing the Apply key and correction of Apan using line 9 (APF variable) and not apan using line 18 (APFWQ variable).
140. In conjunction with the above, on 15 September 2015, the new code for graph enhancements (done by Mr Grant Nyland) was also brought into play using WINT\_graph.
141. Changes were made akb16Oct2015 to wrsm2001.f90 relating to the \*\*\* Obs in the Plot menu for observed routes to use external route numbering and not internal route numbering.
142. Added the graph manual of Grant Nyland to the help. Needed to go into resource.rc and choose Menu Editor and add. Also changed wrsm2001.f90.





9781431208494