CHAPTER 9
PROGRAMMER DOCUMENTATION

This chapter provides code documentation for programmers. This chapter starts with sections on overall design decisions and the use of Fortran modules for sharing data. The remaining sections describe the MAIN Program, each package, and the utility subroutines; the extent of the documentation varies substantially among the sections. The most detailed documentation is provided for the MAIN Program, the Basic (BAS) Package, the Layer-Property Flow (LPF) Package, the River (RIV) Package, Recharge (RCH) Package, and the Utility subroutines. The remaining stress packages are similar to RIV and RCH, so the subroutines are only briefly summarized. The Block-Centered Flow (BCF) Package is similar to LPF. Solvers are only briefly described because rarely are they modified, and programming details can be found in the references. Many details about the programs also are included in the code as Fortran comments, which must be examined to gain total understanding of the code.

Overall Design Decisions

The following paragraphs provide information about a variety of programming decisions or conventions that were adopted for MODFLOW. The coding decisions are not absolute, and coding style is not strictly enforced; however, anyone who examines the code is likely to notice some aspects of the coding decisions. Developers of new code are encouraged to follow the coding decisions.

MODFLOW is written as a batch program, which means that the program is designed to run without user intervention. The primary reason for this is the expectation that execution time will be lengthy. Execution time of course varies with the problem and the power of the computer, but this expectation continues to be valid. Simulations frequently take many minutes, and simulations taking hours or days are common.

The input data are read from files that must be prepared in advance. Although interactive computer programs usually are used to prepare the input data, interactive data input has been intentionally avoided within MODFLOW because this is in conflict with the view that MODFLOW is a batch program. The methods of interactive input are inherently less standard among kinds of computers so that including interactive input would result in a less portable program. The variety of user preferences would also promote the addition of numerous input options that would cause the code to be more difficult to understand. From the perspective of maintaining a compact, efficient, and easily understood code, input data read from files is best.

To maximize computational efficiency and minimize code complexity, MODFLOW includes minimal error checking. Also, when an error is found, the program stops rather than prompting the user to correct the error and resuming computations. This does not mean that the author views error checking as unimportant. Rather, the author believes that the error checking is best done as part of an interactive program that prepares the input files. Having separate data preparation and data checking makes possible the accommodation of the needs and preferences of a variety of users while keeping the model code straightforward and efficient.

Variable names generally are six characters or less, which was originally a requirement of Fortran. Short names also help to keep the code compact. Longer variable names are used sparingly in new parts of the code. Changing the original variable names to longer names at this time would be unwise because of possible confusion.

Many programmers have strong feelings about the issue of specifying the data type of variables. Default implicit typing generally is used in MODFLOW. This means that type statements are not used unless specifically needed; and therefore, the type will be Real for variables starting with letters A–H and O–Z, and the type will be Integer for variables starting with I–N. The default types allow the code to be shorter, but limit flexibility for naming variables. Requiring data types to be declared for all variables also would have the advantage that the compiler would produce an error if an untyped variable were used, which is helpful for avoiding misspelling of variables. Nevertheless, the desire for short, concise code led to the decision to use default typing.

Logical variables generally are avoided. Instead, Integer flags are used in which false is a value of 0 and true is a value of not 0 (usually 1). This is partly evolutionary because the original authors experienced errors with early

Fortran compilers regarding logical variables. Some code simplicity also is gained from avoiding the need for logical type statements and logical operators.

The precision used to represent numbers in the computer determines in large part the accuracy of the numeric solution of the flow equation. When using Fortran, precision depends on the computer hardware and the declared precision of Real numbers. Although the Fortran standard does not directly specify precision requirements, Fortran includes single and double precision Real numbers without specifying the actual precision of either type. Fortran 90 provides a further mechanism for finding out the precision of numbers and specifying what precision is desired, but still no absolute requirements for numeric precision are incorporated. MODFLOW was developed assuming that single precision Real numbers are represented with approximately 32 binary digits and double precision Real numbers are represented with 64, which was and continues (2005) to be common among computers. The default implicit Real type is single precision. When double precision is desired, the type is explicitly declared to be double precision.

Tests of the original MODFLOW code were done to determine the need or benefit from making some variables double precision. The tests showed that a wide variety of problems worked well when using a limited amount of double precision. Head and some of the solver arrays were made double precision based on this testing. Some problems benefit from the use of full double precision, and in general, the larger the problem, the more likely that full double precision is beneficial. Other reasons why double precision might be needed are for high-precision answers and for problems where the head distribution is relatively flat in large areas of the model.

The number of cells in a typical model has increased over the years as the complexity of problems and computer speed have increased; thus, the need for full double precision is greater now than when MODFLOW was originally released (1984). This would argue in favor of eliminating the mixed precision coding in favor of full double precision; however, the original mixed precision continues to be adequate for a variety of problems. There is a penalty of computer memory, disk space, and sometimes on execution time when full double precision is used. The author is reluctant to impose this penalty unnecessarily. Therefore, the original mixed precision code has been maintained in MODFLOW–2005. Modern Fortran compilers have an option to convert all Real numbers to double precision without the need to modify the code, so on most computers, making any code changes to convert the code to double precision is unnecesssary. Simply recompile by using the double precision compiler option. When necessary, code modification to convert MODFLOW to all double precision as described by McDonald and Harbaugh (1988, Appendix A) is still a relatively simple task.

To avoid inconsistent results among early Fortran compilers, mixed precision in a single arithmetic expression was avoided when MODFLOW was developed, and this practice continues. If any operand in an arithmetic expression is double precision, all of the single precision operands are replaced with temporary double precision variables that are set equal to the single precision values. These temporary variables have no affect when converted to full double precision. When constants are needed in arithmetic statements, ambiguity over the precision of the constants is generally avoided by using variables to store the constant values. Regardless of the precision of the constant, the Fortran compiler will cause the constant to be converted to the precision of the variable. The constant value is then referenced using the variable. For example, variables ONE and ZERO are typically used to store values for 1.0 and 0.0, respectively.

Explicit DO Loops are used throughout much of MODFLOW to make assignments of array elements an element at a time; however, in some places a Fortran 90 array assignment statement is used to assign values to all elements of an array without using a Do Loop. The lack of array assignment statements is primarily evolutionary in that this capability was not available until Fortran 90. Further, the author sees benefit in maintaining explicit loops for the sake of clarity. Although array assignment statements result in shorter code, they tend to make the code more obscure. When an array assignment statement is used, one must recognize that a variable is an array to understand what is going on in the code.

Experience has shown that some specifiers used in the Fortran OPEN statement are not standard among all compilers. File openspec.inc contains variable definitions used for opening files. Variables ACCESS, FORM, and ACTION are used as values for the ACCESS, FORM, and ACTION specifiers in open statements. Openspec.inc is incorporated in the code whenever the variables are needed by using a Fortran INCLUDE statement. The values of variables can be modified in openspec.inc and the code recompiled if different values for the open specifiers are required.

Output to the Listing File is mostly written using lines of 80 characters or less. Although computer displays are no longer limited to 80 characters as in the past, a width of 80 characters is judged to be a convenient size for a wide range of uses. For compatibility with older versions of MODFLOW, the utility subroutines continue to support the writing of lines that are 132 characters wide.

Each stress and internal flow package must incorporate a budget subroutine that adds budget data to the VBVL array. The VBVL array is defined as part of Fortran module GWFBASMODULE of the Basic Package. Additional information about the model budget is contained in Chapter 3 of this report. Budget subroutines can optionally support cell-by-cell budgets, and all packages documented in this report support this capability. This capability requires budget data to be written into a file using the appropriate cell-by-cell budget utility subroutine.

When the cell-by-cell budget capability is supported, the budget subroutine should store budget values in array BUFF. BUFF should contain the net inflow for each cell in the entire grid. At cells where IBOUND is 0 or where stress is not applied, BUFF should be set equal to 0. The Ground-Water Flow Process does not use the budget values stored in BUFF, but this is a mechanism for other processes to obtain the budget data. Further, for packages that specify stresses using lists of cell locations (as opposed to layer arrays), the cell-by-cell budget values should be stored in the list of data for each stress location. Again, this can be useful for other processes. Space for this budget data must be reserved in the list of data.

For internal flow packages, flow between adjacent cells is a special kind of cell-by-cell budget data. These data do not represent inflow or outflow for the system as a whole, but the data can be useful for many purposes. These data comprise three terms, one for each coordinate axis as described in Chapter 3. These data are written to a file as is done for inflows and outflows. Flow between adjacent cells is also stored in BUFF for use by the Ground Water Transport Process if the IBDRET flag is not 0. When IBDRET is not 0 and budget data are not written to disk, flow between adjacent cells is computed only for a subset of the grid, which is designated through subroutine arguments. IBDRET is a local variable in the MAIN Program.

Data Declaration and Sharing Using Fortran Modules

Shared data can be declared in Fortran modules. This was not done in the original code because the capability was not available until Fortran 90. Most data were passed to subroutines as arguments, which results in long argument lists, but makes clear which data are used by the subroutines. Shared data in modules are passed to subroutines using the Fortran USE statement. The “ONLY” specifier to USE allows the specific variables being used to be identified in each subroutine. Fortran modules eliminate the need for most subroutine arguments.

Figure 9–1 is an example Fortran module for a simple river package. This is simpler than the RIV Package in MODFLOW–2005, but illustrates the concept of sharing data using a Fortran module. Integer NRIVER is the number of river cells in use in the current time step, and MXRIVR is the maximum number of river cells that can be used in any stress period. RIVR is a two-dimensional array declared as a pointer. RIVR is allocated with the first index being the number of values needed for each river cell, and the second index being MXRIVR. The specific dimensions for RIVR will be declared when the array is allocated in the Allocate and Read Procedure subroutine (GWF2RIV7AR). All of these variables have the SAVE attribute so that they will never become unassociated.

```
MODULE GWFRIVMODULE
  INTEGER,SAVE                              ::NRIVER,MXRIVR
  REAL,   SAVE, DIMENSION(:,:), POINTER     ::RIVR
END MODULE GWFRIVMODULE
```

Figure 9–1. Fortran module for declaring shared data.

The data in the module can be accessed in any subroutine by including the statement

```
USE GWFRIVMODULE,   ONLY:NRIVER,MXRIVR,RIVR
```

If each river cell requires six data values, the memory for RIVR can be allocated with the statement:

```
ALLOCATE (RIVR(6,MXRIVR))
```

A new complexity is introduced when the possibility of Local Grid Refinement (LGR) (Mehl and Hill, 2004) is added to MODFLOW. Although LGR is not described here, the impact on shared data is described. LGR requires data to be defined for multiple grids. At a minimum, a regional grid for the entire modeled area and a refined grid containing a subset of the modeled area is included. The regional and refined grids must both have data defined. In MODFLOW–2005, multiple grids are accommodated by using a derived data type. The derived type contains data for one grid, and an array of the derived type can then store data for multiple grids.

Figure 9–2 is the Fortran module for the RIV Package of MODFLOW–2005, which supports LGR. GWFRIVTYPE is a derived type that includes a complete set of pointers for all data required for a grid. GWFRIVDAT is an array of this type. Thus each element of GWFRIVDAT can contain data for one grid. The data for different grids can be accessed by specifying an element of GWFRIVDAT and the variable within GWFRIVTYPE. For example, the value of MXRIVR for the second grid would be GWFRIVDAT(2)%MXRIVR.

```
MODULE GWFRIVMODULE
TYPE GWFRIVTYPE
  INTEGER,POINTER  ::NRIVER,MXRIVR,NRIVVL,IRIVCB,IPRRIV
  INTEGER,POINTER  ::NPRIV,IRIVPB,NNPRIV
  CHARACTER(LEN=16), DIMENSION(:),   POINTER     ::RIVAUX
  REAL,              DIMENSION(:,:), POINTER     ::RIVR
END TYPE
TYPE(GWFRIVTYPE), SAVE:: GWFRIVDAT(10)
END MODULE GWFRIVMODULE
```

Figure 9–2. Fortran module for declaring shared data with support for multiple grids.

Several additional variables compared with those used for the simple river package (fig. 9–1) also have been included in the module shown in figure 9–2. These variables support some of the optional capabilities of the River Package. See the documentation for the River Package for definitions of all the variables.

Variables in a derived type can be referenced more simply through the use of pointers. Although some extra steps are needed to setup the pointers, the code then looks more like the code without the LGR capability. This approach is implemented by creating pointers that have the same names as the variables in the derived type as shown in figure 9–3. The 4 lines preceding the derived type declaration are the declarations for these variables. Each pointer can be made to point to a variable for a specific grid. The pointers can then be used to access the variables for that grid without repeating the grid designation. For example, the statement

MXRIVR=>GWFRIVDAT(2)%MXRIVR

causes MXRIVR to point to the value of MXRIVR in the derived type for grid 2, and accordingly, GWFRIVDAT(2)%MXRIVR can be accessed simply as MXRIVR.

```
MODULE GWFRIVMODULE
  INTEGER,SAVE,POINTER  ::NRIVER,MXRIVR,NRIVVL,IRIVCB,IPRRIV
  INTEGER,SAVE,POINTER  ::NPRIV,IRIVPB,NNPRIV
  CHARACTER(LEN=16),SAVE, DIMENSION(:),   POINTER     ::RIVAUX
  REAL,             SAVE, DIMENSION(:,:), POINTER     ::RIVR
TYPE GWFRIVTYPE
  INTEGER,POINTER  ::NRIVER,MXRIVR,NRIVVL,IRIVCB,IPRRIV
  INTEGER,POINTER  ::NPRIV,IRIVPB,NNPRIV
  CHARACTER(LEN=16), DIMENSION(:),   POINTER     ::RIVAUX
  REAL,              DIMENSION(:,:), POINTER     ::RIVR
END TYPE
TYPE(GWFRIVTYPE), SAVE:: GWFRIVDAT(10)
END MODULE GWFRIVMODULE
```

Figure 9–3. Fortran module for declaring shared data with support for multiple grids and pointers for simplified access.

To make possible the use of the pointers as simple variables, several steps are needed in addition to declaring the pointers in the module. First, the non-array pointers must be allocated much like the array pointers are allocated. Allocating a scalar pointer causes memory to be created for that scalar. This is done by ALLOCATE statements at the beginning of the Allocate and Read (AR) Procedure subroutine. When LGR is used, the AR subroutine will be called for each grid. At the end of the AR subroutine, the pointers to the variables for the current grid must be saved in the corresponding pointers of the derived type (GWFRIVDAT) for the grid. For example, the statement

```
GWFRIVDAT(2)%MXRIVR=>MXRIVR
```

causes the MXRIVR pointer for the current grid to be saved in the derived type for grid 2. The complete set of these pointer saving statements are placed in a pointer saving subroutine, named SGWF2RIV7PSV for the River Package, which is called at the end of the AR subroutine. All other subroutines that use the data need to insure that the pointers are set to the current grid by setting the simple variable pointers. This is done by calling a pointer setting subroutine, named SGWF2RIV7PNT for the River Package, at the beginning of each subroutine. The pointer setting subroutine contains statements setting the pointers for all the data for the package. Each package that supports LGR will have a pointer saving secondary subroutine and a pointer setting secondary subroutine. These pointer saving and setting subroutines are not specifically documented in the subsequent sections that document each package because they all have the same simple structure.

Fortran provides a mechanism for releasing allocated memory, which is called deallocation. Memory is used throughout a simulation in MODFLOW, so deallocation is not needed until the end. Further, a Fortran program automatically releases all memory when the program terminates, and accordingly, explicit memory deallocation generally is not required in MODFLOW. A deallocation procedure is included in the MODFLOW–2005 flowchart (fig. 3–1), however, and each package includes a deallocation subroutine that is called to release memory from the Ground-Water Flow Process (GWF). The deallocation subroutines are not specifically documented in the subsequent sections that document each package because they all have the same simple structure.

MAIN Program

The MAIN Program controls the order in which the primary subroutines are executed. This occurs with CALL statements that specify by name the subroutines to be executed. The arrangement of CALL statements in the MAIN Program reflects the order of procedures shown in the system flow chart (fig. 3–1). Within most procedures, the calls to the primary subroutines should begin with the Basic Package (if included in that procedure) followed by calls to the other packages in any order. The one exception is that the Basic Package Deallocate subroutine should be the last call in the Deallocate procedure. The reason for this is that the data allocated in the Basic Package may be needed by other deallocate subroutines.

The subroutines of a package are called only if the package is being used in a simulation. The Name File indicates which packages are being used. Each primary option has a unique file type; when the option is a package, the file type is simply the package abbreviation. The defined file types are stored in the one-dimensional CUNIT variable; a DATA statement in the MAIN Program defines these values. CUNIT contains 100 elements, and each element consists of four characters. Unused elements are filled with blanks. A corresponding IUNIT integer variable also has 100 elements. The Basic Package initializes all IUNIT values to 0. When the Name File is read, each file is tested to see if its File Type parameter matches one of the defined file types in CUNIT. If a match is indicated, the unit number for that file is placed in the element of IUNIT that corresponds to the matched element of CUNIT. IUNIT is an indicator of which options are active, and therefore is tested to determine whether or not the subroutines of a package should be called. For example, the Recharge Package corresponds to the eighth element of IUNIT. If IUNIT(8) is greater than 0; then the Recharge Package is active, and the primary subroutines of the Recharge Package are all called within the appropriate procedures.

The MAIN Program is the only place where Fortran modules are used without the “ONLY” option. The reason for using modules in the MAIN Program is to allow all data to pass to the solvers using subroutine arguments. The subroutine arguments indicate which variables are used.

File “openspec.inc” appears in an Include Statement in the MAIN Program because openspec.inc defines file attributes that can sometimes vary among compilers. Putting such attributes in a separate file allows the code to be changed in many places by modifying the single file. In the MAIN Program, the Open Statement for the Name File uses one of the file attributes (ACTION).

Character variable VERSION is defined using a Fortran Parameter Statement. VERSION is used to identify the specific version of MODFLOW.

The executable part of the code starts by writing VERSION to the standard output. Then, two unit numbers are defined. INUNIT is the unit number for the Name File, and IERRU is the output unit number for writing errors.

The MAIN Program makes use of a special secondary subroutine, GETNAMFIL, to get the name of the Name File. This subroutine is not part of any MODFLOW package. Isolating this code in a separate subroutine makes the MAIN Program easier to follow and makes modifying the way the Name File is defined easier should this be desired. GETNAMFIL first attempts to get the Name File from the command line. If the command line argument is absent, the user is prompted to enter a file name. Modifying the call to a system subroutine that retrieves the command line may be necessary. After calling GETNAMFIL, the MAIN Program opens the Name File and writes the name to standard output.

The next step in MAIN is to call the standard Fortran subroutine for obtaining the date and time. This will be used at the end of the program to compute the elapsed simulation time.

The variable IGRID is incorporated in some primary subroutines to facilitate the addition of the Local Grid Refinement (LGR) capability mentioned above. Variable IGRID is set equal to 1 in MAIN, which indicates the primary grid.

The loop indices for iteration, time steps, and stress periods are needed in many subroutines; however, these are not directly passed as arguments. Rather, duplicate variables are assigned the values and the duplicates are passed. This avoids the appearance that these loop indices could be modified, which allows some Fortran compilers to perform greater optimization.

Like subroutine GETNAMFIL, subroutine GLO1BAS7ET is a secondary subroutine of MAIN. GLO1BAS7ET is used to compute execution time for the simulation and write the execution time to the standard output. This is an unessential part of the program that users may wish to modify or delete. This routine makes a second call to the standard Fortran date and time routine and computes the elapsed time since the initial call.

The following is a summary of the MAIN Program (the numbers in the list correspond to the numbers of the comments in the MAIN Program listing):

1. USE package modules for purpose of data sharing.

2. Write banner to screen and assign 99 as the input unit number for the Name File.

3. Call GETNAMFIL to obtain the name of the Name File either from a command argument or from a user response to a prompt. This code was placed in a subroutine rather than directly in the MAIN to avoid obscuring the structure of the MAIN.

4. Open the Name File.

5. Get current date and time so execution time can be tracked.

6. Call primary subroutines in the AR Procedure. (GWF1BAS6AR reads the Name File and assigns IUNIT values.)

7. For each stress period:
    - 7A. Read stress-period timing information.
    - 7B. Read and prepare information that changes each stress period.
    - 7C. For each time step:
        - 7C1. Calculate the current time-step length and move "new" heads from the preceding time step to the variable containing "old" heads of the current time step.

7C2. For each solution iteration:
    7C2A. Formulate the finite-difference equations.
    7C2B. Calculate an approximate solution to the system of equations.
    7C2C. If convergence criterion has been met, stop iterating.
7C3. Determine the type and amount of output needed for this time step.
7C4. Calculate overall budget terms and, if specified, calculate and print or record cell-by-cell flow terms.
7C5 Print and/or record heads and/or drawdown. Print the overall volumetric budget and simulation time summary.
7C6. If convergence criterion was not met during iteration, STOP.

8. Call GLO1BAS7ET to get date and time at end of simulation.

9. Close files and deallocate memory.

10. End of program.

Basic Package

Although the Basic (BAS) Package does not contribute terms to or solve the flow equation, the BAS Package performs many logistical functions. BAS declares and allocates memory for variables that can be shared throughout the GWF and other processes, initializes variables that are used to construct the flow equation, reads the Name File and opens files, reads discretization data, and implements output control.

Basic Package Data

The shared Basic Package variables are declared in four Fortran modules: GLOBAL, PARAMMODULE, GWFBASMODULE, and GWFCHDMODULE; tables 9–1, 9–2, 9–3, and 9–4 define these variables, respectively. Some of the program variables have different names than the variables used to develop the finite-difference equation in Chapter 2, and these differences are described in the tables.

GLOBAL Module

NIUNIT, which defines the size of the IUNIT, VBVL, and VBNM arrays, is defined using a Fortran Parameter statement. These arrays cannot be readily defined on the basis of user requirements because these arrays are allocated prior to the user specifying information about the required array dimensions.

PARAMMODULE Module

Parameter definitions for all packages are stored in a group of variables contained in Fortran module PARAMMODULE. The parameter variables are somewhat complex, and it is recommended that they be accessed only through the parameter utility subroutines. These routines create parameters and access all of the associated data.

The primary parameter data are stored in one-dimensional arrays. PARNAM contains the parameter name, PARTYP contains the parameter type, B contains the parameter value, and IACTIVE is a flag indicating whether a parameter is active in the current time step. The parameters are stored in the order in which they are defined. Parameters are found when needed by a simple sequential search through PARNAM.

Parameter indexing data depend on whether a parameter is an array parameter or a list parameter. An array parameter requires a set of clusters that define the cells in each layer that are associated with the parameter. Clusters are stored in IPCLST. Pointers to the first and last cluster for a parameter are stored in IPLOC. For example, if parameter 7 is a hydraulic conductivity parameter, then IPLOC(1,7) and IPLOC(2,7) contain the location of the first and last clusters in IPCLST associated with this parameter.

A list parameter requires pointers to the first and last cells associated with the parameter in the list that stores data for the package. These pointers are stored in IPLOC. For example, if parameter 5 is a river parameter, then IPLOC(1,5) and IPLOC(2,5) contain the location in RIVR of the first and last cells associated with this parameter.

The capability to have multiple instances of the same parameter adds further complexity to parameter definition. This capability is used to allow parameter definitions to change during the simulation. Each instance has the same parameter value, but the cells associated with the parameter can be different. Thus, each array parameter instance has its own clusters, and each list parameter instance has its own cell list. IPLOC(3,n) indicates the number of instances. If there are instances for an array parameter, then IPLOC(2,n) is the last cluster of the last instance. Each instance must have the same number of clusters. Similarly, if there are instances for a list parameter, IPLOC(2,n) is the location of the last cell of the last instance. Each instance must have the same number of cells.

Several variables that specify array dimensions (MXPAR, MXCLST, and MXINST) are defined using Fortran Parameter statements. These cannot be readily computed from input data because the arrays are allocated prior to the user specifying information about the array dimensions.

Table 9–1. Variables in Fortran module GLOBAL.

Variable Name	Size	Description
NIUNIT	**Scalar**	**The dimension of the IUNIT, VBVL, and VBNM arrays. Initially set equal to 100 using a Parameter statement.**
NCOL	**Scalar**	**The number of columns in the model grid.**
NROW	**Scalar**	**The number of rows in the model grid.**
NLAY	**Scalar**	**The number of layers in the model grid.**
NPER	**Scalar**	**The number of stress periods in the simulation.**
NBOTM	**Scalar**	**The number of layers of data in the BOTM array.**
NCNFBD	**Scalar**	**The number of model layers that are underlain by Quasi–3D confining beds.**
ITMUNI	**Scalar**	**Time unit code: 0=undefined, 1=seconds, 2=minutes, 3=hours, 4=days, 5=years**
LENUNI	**Scalar**	**Length unit code: 0=undefined, 1=feet, 2=meters, 3=centimeters**
IXSEC	**Scalar**	**Cross section flag that is set to 1 if “XSECTION” is found in the option line of Basic Package file: 0 indicates not a cross section, 1 indicates a 1-row cross section.**
ITRSS	**Scalar**	**Flag: 0=steady state, 1=transient, -1=combined steady state and transient.**
INBAS	**Scalar**	**File unit number for the BAS Package of the GWF Process.**
IFREFM	**Scalar**	**Free format flag that is set to 1 if “FREE” is found in the option line of Basic Package file: 0=fixed format, 1=free format.**
NODES	**Scalar**	**The number of nodes (or cells) in the model grid.**
IOUT	**Scalar**	**The file unit number for the Listing File.**
IUNIT	**NIUNIT**	**An array of unit numbers. Each package or option that requires an input file is assigned an element in IUNIT. If that element is 0, the package or option is inactive. If the element is greater than 0, the package or option is active and the value is unit number for reading data.**
LBOTM	**NLAY**	**Array of pointers to the 3rd index of the BOTM array. For example, LBOTM(2) is the index in BOTM of the bottom of layer 2.**
LAYCBD	**NLAY**	**Quasi–3D confining bed flag for each layer: 0 indicates no confining bed under a layer, not 0 indicates a confining bed.**
LAYHDT	**NLAY**	**Head-dependent transmissivity flag: 0 indicates transmissivity for a layer is constant, 1 indicates transmissivity varies with head.**
LAYHDS	**NLAY**	**Head-dependent storage flag: 0 indicates storage for a layer is constant, 1 indicates storage varies with head.**
PERLEN	**NPER**	**The length of a stress period.**
NSTP	**NPER**	**The number of time steps in a stress period.**
TSMULT	**NPER**	**Multiplier for the length of successive time steps in a stress period.**
ISSFLG	**NPER**	**Steady State/Transient flag for a stress period – “SS” for steady state and “TR” for transient.**
DELR	**NCOL**	**The cell width along rows (Δr in equations used throughout this report). DELR(J) is the width of column J.**
DELC	**NROW**	**The cell width along columns (designated as Δc in equations throughout this report). DELC(I) is the width of row I.**
BOTM	**NCOL,NROW,0:NBOTM**	**Elevation of cell and Quasi–3D confining bed bottoms. A value of 0 for the 3rd index is used to store the system top elevation.**
HNEW	**NCOL,NROW,NLAY**	**Computed head at the current time step (designated as h^m throughout this report).**
HOLD	**NCOL,NROW,NLAY**	**Head from the end of the previous time step (designated as h^{m-1} throughout this report).**
IBOUND	**NCOL,NROW,NLAY**	**Boundary code: <0 – specified head, =0 – no flow, >0 – variable head.**
CR	**NCOL,NROW,NLAY**	**Conductance along rows. CR(J,I,K) is the conductance between cell (J,I,K) and (J+1,I,K), where J is the column index.**
CC	**NCOL,NROW,NLAY**	**Conductance along columns. CC(J,I,K) is the conductance between cell (J,I,K) and (J,I+1,K), where I is the row index.**
CV	**NCOL,NROW,NLAY-1**	**Vertical conductance. CV(J,I,K) is the conductance between cell (J,I,K) and (J,I,K+1), where K is the column index.**
HCOF	**NCOL,NROW,NLAY**	**Coefficient of head in ground-water flow equation.**
RHS	**NCOL,NROW,NLAY**	**Right-hand side of ground-water flow equation.**
BUFF	**NCOL,NROW,NLAY**	**Temporary buffer for use within a subroutine.**
STRT	**NCOL,NROW,NLAY**	**Initial head.**

Table 9–2. Variables in Fortran module PARAMMODULE.

[“C*n” in size column indicates a character variable of n characters]

Variable Name	Size	Description
MXPAR	Scalar	Maximum number of parameters allowed. Initially set equal to 500 using a Parameter statement.
MXCLST	Scalar	Maximum number of clusters that can be used for defining array parameters. Initially set equal to 1000 using a Parameter statement.
MXINST	Scalar	Maximum number of zone arrays. Initially set equal to 1000 using a Parameter statement.
ICLSUM	Scalar	The number of defined array parameter clusters.
IPSUM	Scalar	The number of defined parameters.
INAMLOC	Scalar	Pointer to the next unused instance name.
NMLTAR	Scalar	The number of multiplier arrays
NZONAR	Scalar	The number of zone arrays.
NPVAL	Scalar	The number of parameters for which values are defined in the Parameter Value File.
B	MXPAR	Parameter values.
IACTIVE	MXPAR	Active parameter flags: -1 – Active in all time steps. 0 – Not active in current time step. >0 – Active in current time step – if using instances, IACTIVE is the instance number.
IPLOC	4,MXPAR	Parameter index. Values of IPLOC(n,p), where p is the parameter number: n=1 – First cluster or list location of first instance. n=2 – Last cluster or list location of last instance. n=3 – Number of instances. n=4 – Location in INAME of first instance name.

IPCLST	14,MXCLST	IPCLST(n,c) – c=cluster number:
		n=1 – Layer number.
		n=2 – Multiplier array number (0 indicates none).
		n=3 – Zone array number (0 indicates all cells).
		n=4 – Index of last zone number for this cluster.
		n=5-14 – Zone numbers.

PARNAM	C*10,MXPAR	Parameter names.
PARTYP	C*4,MXPAR	Parameter types.
ZONNAM	C*10,MXZON	Zone array names.
MLTNAM	C*10,MXMLT	Multiplier array names.
INAME	C*10,MXINST	Instance names.
RMLT	NCOL,NROW,NMLTAR	Multiplier arrays for defining parameters.
IZON	NCOL,NROW,NZONAR	Zone arrays for defining parameters.

GWFBASMODULE Module

Table 9–3. Variables in Fortran module GWFBASMODULE.

[“C*n” in size column indicates a character variable of n characters]

Variable Name	Size	Description
MSUM	**Scalar**	**Budget term counter.**
IHEDFM	**Scalar**	**Format code for writing head in the Listing File.**
IHEDUN	**Scalar**	**File unit number for saving head in a file.**
IDDNFM	**Scalar**	**Format code for writing drawdown in the Listing File.**
IDDNUN	**Scalar**	**File unit number for saving drawdown in a file.**
IBOUUN	**Scalar**	**File unit number for saving IBOUND in a file.**
LBHDSV	**Scalar**	**Label flag for saving head in a formatted file:** **0 –Do not write label.** **not 0 – Write label.**
LBDDSV	**Scalar**	**Label flag for saving drawdown in a formatted file:** **0 – Do not write label.** **not 0 – Write label.**
LBBOSV	**Scalar**	**Label flag for saving IBOUND in a formatted file:** **0 – Do not write label.** **not 0 – Write label.**
IBUDFL	**Scalar**	**Flag for writing overall budget in the Listing File in current time step:** **0 – Do not write budget.** **not 0 – Write budget.**
ICBCFL	**Scalar**	**Flag for writing cell-by-cell budget data in current time step:** **0 – Do not write data.** **1 – Write data using noncompact form.** **2 – Write data using compact form.**
IHDDFL	**Scalar**	**Flag for output of head and drawdown in current time step:** **0 – No output.** **1 – Write output according to IOFLG.**
IAUXSV	**Scalar**	**Auxiliary flag for saving cell-by-cell budget data:** **0 – Do not save auxiliary data in budget file.** **not 0 – Save auxiliary data in budget file.**
IBDOPT	**Scalar**	**Compact budget option:** **1 – Noncompact** **2 – Compact**
IPRTIM	**Scalar**	**A flag that is set to 1 if “PRINTTIME” is specified in the option line of the Basic Package file. This is used to determine if execution time is written to the Listing File.**
IPEROC	**Scalar**	**The stress period at which the next output control should occur when using alphabetic output control. If numeric output control is used, the value is -1.**
ITSOC	**Scalar**	**The time step at which the next output control should occur when using alphabetic output control. If numeric output control is used, the value is -1.**
ICHFLG	**Scalar**	**A flag that is set to 1 if “CHTOCH” is specified in the option line of the Basic Package file. This indicates that flow between constant-head cells should be computed when cell-by-cell budget terms are computed.**
DELT	**Scalar**	**Length of current time step. (Designated as $\Delta t = t^{m}-t^{m-1}$ throughout this report.)**
PERTIM	**Scalar**	**Total simulation time in current stress period.**
TOTIM	**Scalar**	**Total simulation time.**
HNOFLO	**Scalar**	**Value substituted in HNEW at no-flow cells.**
CHEDFM	**C*20**	**Format for saving head in a file (blank indicates unformatted).**
CDDNFM	**C*20**	**Format for saving drawdown in a file (blank indicates unformatted).**
CBOUFM	**C*20**	**Format for saving IBOUND in a file.**
IOFLG	**NLAY,5**	**Flags for output of data in current time step (k is a layer index):** **(k,1) Not 0 – Write head to Listing File.** **(k,2) Not 0 – Write drawdown to Listing File.** **(k,3) Not 0 – Save head to unit IHEDUN.** **(k,4) Not 0 – Save drawdown to unit IDDNUN.** **(k,5) Not 0 – Save IBOUND to unit IBOUUN.**

VBVL	4,NIUNIT	Overall budget values (n is a budget term index): (1,n) – Inflow rate for current time step. (2,n) – Outflow rate for current time step. (3,n) – Cumulative volume of inflow. (4,n) – Cumulative volume of outflow.
VBNM	C*16,NIUNIT	Names of budget terms.

GWFCHDMODULE Module

Table 9–4. Variables in Fortran module GWFCHDMODULE.

[“C*n” in size column indicates a character variable of n characters]

Variable Name	Size	Description
NCHDS	Scalar	The number of constant-head cells in the current stress period.
MXCHD	Scalar	The second dimension of CHDS, which includes space for the active constant-head cells in a stress period and all constant-head cells defined by parameters.
NCHDVL	Scalar	The number of the first dimension of CHDS array, which includes five input values and the auxiliary data.
IPRCHD	Scalar	Flag for printing constant-head data – 0 indicates do not print, 1 indicates print.
NPCHD	Scalar	The number of constant-head parameters.
ICHDPB	Scalar	The value of the second index in CHDS at which parameter data begins.
NNPCHD	Scalar	The number of nonparameter constant-head cells in the current stress period.
CHDAUX	C*16,20	The name of auxiliary variables.
CHDS	NCHDVL,MXCHD	Constant-head list, which includes space for the constant-head cells active in a stress period and all constant-head cells defined by parameters. CHDS does not include constant-head cells specified through the initial value of IBOUND.

Subroutines

The Basic Package contains six primary subroutines: GWF2BAS7AR, GWF2BAS7ST, GWF2BAS7AD, GWF2BAS7FM, GWF2BAS7OC, and GWF2BAS7OT. The subroutines GWF2BAS7AR, GWF2BAS7OC, and GWF2BAS7OT each make use of secondary subroutines to reduce the size of the primary subroutines. The Time-Variant Specified-Head (CHD) Option further consists of subroutines GWF2CHD7AR, GWF2CHD7RP, and GWF2CHD7AD. These CHD subroutines are named as if they are a package because this code was originally viewed as a package (Leake and Prudic, 1991). The CHD subroutines also are called from the MAIN Program as if they are primary subroutines. Nevertheless, as mentioned in Chapter 4, CHD is viewed functionally as an option within the BAS Package because of its function to define IBOUND values.

GWF2BAS7AR

This subroutine allocates memory and reads data for the Basic (BAS) Package. The BAS Package file is read directly by this subroutine. The Name, Discretization, Output Control, Zone, and Multiplier files are read by secondary subroutines called by GWF2BAS7AR.

Arguments:

INUNIT – Input unit number for the Name File
CUNIT – Array of file types for primary options
VERSION – MODFLOW version as a text string
IUDIS – Element in IUNIT that is used for the Discretization File
IUZON – Element in IUNIT that is used for the Zone File
IUMLT – Element in IUNIT that is used for the Multiplier File
MAXUNIT – The maximum unit number used in the Name File
IGRID – The grid number, which is used for local grid refinement
IUOC – Element in IUNIT that is used for the Output Control File
HEADNG – A two-line simulation title
IUPVAL – Element in IUNIT that is used for the Parameter Value File

The AR subroutine performs its work in the following sequence:

1. Allocate the scalar variables.
2. Call secondary subroutine to open files.
3. Identify package and initialize data.
4. Initialize parameter definition variables.
5. Call a secondary subroutine to allocate memory for discretization and read the discretization file.
6. Allocate remaining global data.
7. Initialize LAYHDT and LAYHDS to -1. These are flags for each layer indicating if transmissivity and storage are head dependent. These flags should be defined by the internal flow package so they can be used by other packages. LAYHDT=0 indicates transmissivity for a layer is constant; LAYHDT=1 indicates transmissivity varies with head. LAYHDS=0 indicates storage for a layer is constant; LAYHDS=1 indicates storage varies with head.
8. Read the Basic Package file.

   8A. Read the comments, saving the first two in variable HEADNG.

   8B. Look for options in the first item after the heading.

   8C. Print the options.

   8D. Initialize TOTIM to 0.0.

   8E. Read boundary array. If the cross-section option is in effect, read the data as a single array. If not a cross section, read the data a layer at a time.

   8F. Read and print HNOFLO, which is the value to be printed at cells where IBOUND is initially 0.

   8G. Read initial heads into STRT as multiple layer arrays or as a single array for a cross section.
9. Copy initial heads from STRT to HNEW. If IBOUND is zero, set HNEW equal to the double precision equivalent of HNOFLO.
10. Call a secondary subroutine to allocate memory for the Output Control Option and read preliminary data.
11. Initialize volumetric budget accumulators.
12. Call a secondary subroutine to allocate memory for zone and multiplier arrays and read these arrays.
13. Call a secondary subroutine to read parameter values from the Parameter Value File.
14. Save memory pointers for the grid and return.

SGWF2BAS7OPEN

This subroutine reads the Name File and opens all of the files in the Name File. File types in the Name File are compared to values in array CUNIT, which is defined in the MAIN Program. The file unit number for a file is saved in the element of IUNIT that is the same as the element of CUNIT that matches the file type. For example if the file type of a file is “RIV” and element 4 of CUNIT contains “RIV,” then the file unit is saved in IUNIT(4). Thus, IUNIT(4) can be tested for a not 0 value to determine if the River Package is active.

Arguments:

INUNIT – Input unit number for the Name File
IOUT – File unit number for Listing File
IUNIT – Array of file unit numbers for primary options
CUNIT – Array of file types for primary options
NIUNIT – The number of elements in IUNIT
VERSION – MODFLOW version as a text string
INBAS – The input unit number for the Basic Package
MAXUNIT – The maximum unit number used in the Name File

SGWF2BAS7ARDIS

This subroutine allocates and reads discretization data, both space and time.

Arguments:

INUNIT – Input unit number for the Name File
CUNIT – Array of file types for primary options
VERSION – MODFLOW version as a text string
IUDIS – Element in IUNIT that is used for the Discretization File
IUZON – Element in IUNIT that is used for the Zone File
IUMLT – Element in IUNIT that is used for the Multiplier File
MAXUNIT – The maximum unit number used in the Name File
IGRID – The grid number, which is used for local grid refinement
IUOC – Element in IUNIT that is used for the Output Control File
HEADNG – A two-line simulation title

This subroutine performs its work in the following sequence:

1. Check that the input file for discretization has been specified in the Name File. Stop the simulation if the file is not defined.

2. Read the comments and the first line after the comments.

3. Use URWORD to get NLAY, NROW, NCOL, NPER, ITMUNI, and LENUNI from the line.

4. Print NLAY, NROW, and NCOL.

5. Print a message showing the time units.

6. Print a message showing the length units.

7. Allocate the flags that keep track of bottom elevation arrays (LBOTM) and confining bed arrays (LAYCBD).

8. Read confining bed flags into LAYCBD.

9. Loop through the number of layers counting confining beds as indicated by LAYCBD being not 0. Save a pointer to the confining bed number in LAYCBD. LAYCBD(K) will be the layer index for VKCB in the Layer-Property Flow Package. Also, set up LBOTM to contain the third index to the bottom elevation array BOTM for each model layer. Elevation arrays for model layers and confining beds are stored in BOTM in the order of depth. Compute NBOTM, which is the sum of the number of model layers and the number of confining beds.

10. Allocate space discretization arrays (DELR, DELC, and BOTM) and time discretization arrays (PERLEN, NSTP, TSMULT, and ISSFLG). BOTM requires NBOTM+1 layers of data, which are indexed from 0 to NBOTM. Layer 0 is for the top elevation for layer 1, which is the overall top of the simulated system.

11. Read DELR and DELC.
12. Read the top elevation for layer 1.
13. Loop through the model layers reading the bottom elevation for each layer as well as the confining bed bottom elevation for layers that have a confining bed.
14. Read and write data defining the NPER stress periods. For each stress period, a line from the input file contains the length of the stress period, the number of time steps, the time step multiplier, and a code indicating steady state or transient. Use URWORD to get each value. Check that the transient/steady-state code is valid and set a numeric code in ISSFLG: 0 for transient and 1 for steady state. Set ISS to 1 if at least one stress period is steady state, and set ITR to 1 if at least one stress period is transient.
15. Check for invalid stress period data. All stress periods must have at least one time step. Only steady-state stress periods can have 0 length. All time step multipliers must be positive. No stress periods can have negative length.
16. Use the ISS and ITR flags from step 14 to define the value for ITRSS, which is a flag that indicates the overall steady-state/transient status of the simulation: 0 indicates all stress periods are steady state, 1 indicates all stress periods are transient, and -1 indicates a combination of steady-state and transient stress periods.
17. Return.

SGWF2BAS7I

This subroutine sets up the Output Control Option. Arrays IOFLG, VBVL, and VBNM are allocated. If the Output Control Option is not active, default output control is setup. Values are then set in IOFLG to cause head for all layers to be written. GWF2BAS7OC further implements default output by invoking the output of head and overall budget at the end of each stress period.

Arguments:

NLAY – The number of layers in the model grid
INOC – Input unit number for the Name File
IOUT – File unit number for the Listing File
IFREFM – Free format flag:
- 0 – Fixed format
- 1 – Free format

NIUNIT – The number of elements in IUNIT

If the Output Control Option is active, the first record is read from the output control file. The first record is examined to determine whether alphabetic or numeric coding is being used. If alphabetic coding is used, then SGWF2BAS7J is called to set up output control. If numeric coding is used, the values from the first record are read, and IPEROC and ITSOC are set equal to -1 as an indicator of numeric output control.

SGWF2BAS7J

This subroutine sets up output control using alphabetic coding. Lines are read from the output control file until a line starting with “PERIOD” is found. The stress period and time step indicated by this line are saved in variables IPEROC and ITSOC, respectively. This subroutine primarily consists of a sequence of calls to URWORD and “IF” tests to search for the alphabetic commands.

Arguments:

INOC – Input unit number for the Name File
IOUT – File unit number for the Listing File
LINE – Line of text from the Output Control file

LLOC – Location pointer within LINE
ISTART – Location of the start of a word in LINE
ISTOP – Location of the end of a word in LINE

SGWF2BAS7ARMZ

This subroutine allocates and reads zone and multiplier arrays.

Arguments:

INZONE – Input unit number for the Zone File
INMULT – Input unit number for the Multiplier File

SGWF2BAS7ARPVAL

This subroutine reads and stores parameter values from the Parameter Value File. This subroutine creates a parameter for each of the listed parameters, but only the name and value are defined. The other values necessary to define a parameter must be defined in the internal flow or stress package that uses the parameter.

Arguments:

IUPVAL – Element in IUNIT corresponding to the Parameter Value File.

GWF2BAS7ST

This subroutine initializes a new stress period. Using the stress period data from the discretization file, the length of the first time step is computed. PERTIM is set to 0. If the first stress period, GWF2BAS7ST calls GWF2BAS7STPVAL to check that all parameters for which a parameter value was defined in the Parameter Value File have been fully defined.

Arguments:

KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement. Note: Each grid has its own time data; however, this data is required to be the same for all grids.

SGWF2BAS7STPVAL

This subroutine checks that all parameters for which a parameter value was defined in the Parameter Value File have been fully defined. NPVAL contains the number of parameters specified in the Parameter Value File. A blank value for the parameter type indicates an undefined parameter.

Arguments: None

GWF2BAS7AD

This subroutine advances to a new time step of a stress period. The length of the new time step (DELT) is computed unless the new step is step 1. DELT is added to TOTIM and PERTIM. HOLD is set equal to HNEW.

Arguments:

KPER – Current stress period
KSTP – Current time step
IGRID – The grid number, which is used for local grid refinement

GWF2BAS7FM

This subroutine initializes HCOF and RHS to 0 so that the various packages can add terms to them. This is one place where an array assignment statement is used to initialize all elements of an array without an explicit Do Loop.

Arguments:

IGRID – The grid number, which is used for local grid refinement

GWF2BAS7OC

This subroutine is called every time step to set output control flags. If default output control is being used, flags are set to write head (IOFLG) and overall budget (IBUDFL) to the Listing File in the time step of each stress period.

If alphabetic coding of output control is being used, SGWF2BAS7N is called to read output control information. If numeric coding is being used, the output control records are read for the time step. If the time step is the last time step of a stress period, the overall budget flag (IBUDFL) is set even if IBUDFL was not specified in the output control data. This automatic setting of IBUDFL is done to prevent a user from running a model without having any budget output.

Arguments:

KSTP – Current time step
KPER – Current stress period
ICNVG – Convergence flag:
  0 – Not converged
  not 0 – Converged
INOC – Element in IUNIT that is used for the Output Control option
IGRID – The grid number, which is used for local grid refinement

SGWF2BAS7N

This subroutine reads alphabetic output control time step information and sets output flags. The output control information for a time step consists of a record specifying a stress period and time step followed by one or more records indicating the kinds of output. Not all time steps must be specified. When Subroutine SGWF2BAS7I detects alphabetic output control, SGWF2BAS7J is called to read preliminary output control data. Data are read until the first stress period/time step record is found. The specified stress period and time step values are stored in IPEROC and ITSOC.

SGWF2BAS7N is called every time step by GWF2BAS7OC. The output control times must be entered in order of increasing time. The current stress period and time step are compared to IPEROC and ITSOC, and three possibilities exist:

1. The current time can be earlier than IPEROC & ITSOC. If so, output does not occur this time step. The I/O flags are cleared, and no output control records are read.

2. The current time can exactly match IPEROC & ITSOC. This means output flags should be read and acted upon this time step. Lines are read until a new stress period/time step record is found (or end of file). As output records are found, the appropriate flags are set. When a new stress period/time step flag is found, they are saved in IPEROC and ITSOC so that subsequent calls to SGWF2BAS7N can compare them to the current simulation time. If the end of the output control file is found, IPEROC and ITSOC are set to 9999, which is presumably greater than the stress period and time step values for the remainder of the simulation.

3. The current time can be later than the IPEROC & ITSOC time. This can only happen if the stress period and time step values are entered out of order in the output control file or if a nonexistent time step is specified. For example, output control for stress period 1 and time step 2 might be specified after stress period 1 and time step 3. This causes IPEROC and ITSOC to be set equal to KPER and KSTP so that output occurs in the current time step.

Arguments:

KPER – The current stress period
KSTP – The current time step
INOC – Input unit number for the Output Control file
IOUT – File unit number for the Listing File

SGWF2BAS7L

When SGWF2BAS7N determines that head, drawdown, or IBOUND should be written (to the Listing File or a separate file), SGWF2BAS7L is called to determine which layers should be written. By default all layers are written, but a list of layer numbers can be specified at the end of the output control line. URWORD is repeatedly called to retrieve an Integer value. A value outside of the range of model layers terminates the check. Note that specification of no layer numbers at the end of the line results in the first call to URWORD returning 0 for the layer number, which causes all layers to be written.

Arguments:

IPOS – Value of second index of IOFLG for which layers are being specified
LINE – Line of data from the Output Control File
LLOC – Location in LINE
IOFLG – Flags for output of data in current time step
NLAY – Number of layers in the grid
IOUT – File unit number for the Listing File
LABEL – A label specifying the kind of data (head, drawdown, or IBOUND)
INOC – Input unit number for Output Control file

GWF2BAS7OT

This subroutine outputs head, drawdown, IBOUND, and overall budget for the time steps specified by GWF2BAS7OC. These are each output using a separate secondary subroutine. If head, drawdown, or the overall budget are written to the Listing File, IPFLG is set to 1, which causes SGWF2BAS7T to be called to write simulation time to the Listing File.

Arguments:

KSTP – Current time step
KPER – Current stress period
ICNVG – Convergence flag:
  0 – Not converged
  not 0 – Converged
ISA – Equation solution flag:
  0 – Flow equation was not solved this time step.
  not 0 – Flow equation was solved this time step.
IGRID – The grid number, which is used for local grid refinement

SGWF2BAS7H

This subroutine writes head to the Listing File and separate file according to flags in array IOFLG that have been previously set by output control.

Arguments:

KSTP – Current time step
KPER – Current stress period

IPFLG – Flag indicating whether output has been written to the Listing File in current time step:
    0 – There has been no output to Listing File.
    not 0 – There has been some form of output to the Listing File.
ISA – Equation solution flag:
    0 – Flow equation was not solved this time step.
    Not 0 – Flow equation was solved this time step.

SGWF2BAS7D

This subroutine writes drawdown to the Listing File and separate file according to flags in array IOFLG that have been previously set by output control. Drawdown is computed as the difference between initial head (STRT) and the current head (HNEW). Where IBOUND is 0, drawdown is specified to be the value of HNEW, which will either be HNOFLO or HDRY depending on the reason why the cell is dry. If the cell is dry because IBOUND was initially 0, HNEW will be HNOFLO. If the cell is dry because the head dropped below the bottom elevation, HNEW will be HDRY.

Arguments:

KSTP – Current time step
KPER – Current stress period
IPFLG – Flag indicating whether output has been written to the Listing File in current time step:
    0 – There has been no output to Listing File.
    not 0 – There has been some form of output to the Listing File.
ISA – Equation solution flag:
    0 – Flow equation was not solved this time step.
    Not 0 – Flow equation was solved this time step.

SGWF2BAS7IB

This subroutine writes IBOUND to a separate file according to flags in array IOFLG that have been previously set by output control.

Arguments:

KSTP – Current time step
KPER – Current stress period

SGWF2BAS7V

This subroutine writes the overall budget. All of the individual budget terms have been stored in variable VBVL by the packages that compute them, but the totals must still be computed. Budget values in the range from 0.1 to 9.99999E11 (BIGVL1) are printed using a fixed F17.4 format. This makes comparing the magnitude of values easy. Outside of this range, values are printed using an exponential format so that at least five digits are printed. The upper value of the range for using a fixed format for printing the difference between total inflow and outflow is decreased to 9.99999E10 (BIGVL2) because the difference can be negative, which requires an extra space for printing.

Arguments:

MSUM – Budget term counter
VBNM – Names of budget terms
VBVL – Overall budget values
KSTP – Current time step
KPER – Current stress period
IOUT – File unit number for Listing File

SGWF2BAS7T

This subroutine writes three values of simulation time: total time, stress period time, and length of the current time step. This subroutine is called at every time step for which head, drawdown, or overall budget are written to the Listing File. If the time unit is specified (ITMUNI>0), then the three values of time are converted to seconds. Each time is then converted to minutes, hours, days, and years; and all of the times are written in a table to the Listing File. A year is taken to be 365.25 days. If the time unit is not specified, then no conversion is made, and the time is written with the unknown units.

Arguments:

KSTP – Current time step
KPER – Current stress period
DELT – The length of the current time step
PERTIM – Total simulation time in current stress period
TOTIM – Total simulation time
ITMUNI – Time unit code
IOUT – File unit number for Listing File

Time-Variant Specified-Head Option Subroutines

Once a cell is made constant head, the cell stays constant head throughout the remainder of the simulation. The “active” constant-head cells in a stress period are those for which heads are being specified.

GWF2CHD7AR

This subroutine allocates memory for the CHD Option.

Arguments:

IN – Input unit number for the CHD Option
IGRID – The grid number, which is used for local grid refinement

GWF2CHD7RP

This subroutine reads data for the CHD Option every stress period.

Arguments:

IN – Input unit number for the CHD Option
IGRID – The grid number, which is used for local grid refinement

GWF2CHD7AD

This subroutine sets the head for the constant-head cells specified by the CHD Option every time step. Input data read by GWF2CHD7RP specifies the head at the beginning and end of each stress period. GWF2CHD7AD uses linear interpolation to define the head for each time step.

Arguments:

KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

Block-Centered Flow Package

The Block-Centered Flow (BCF) Package computes conductance terms for flow between cells and storage terms. When simulating confined flow, the computations are straightforward, but complexities such as unconfined conditions, cell drying, cell wetting, and conversion between confined and unconfined conditions cause the code to be much more complex. Shared data for the BCF Package are declared in Fortran Module GWFBCFMODULE and defined in table 9–5.

Table 9–5. Variables in Fortran module GWFBCFMODULE.

Variable Name	Size	Description
IBCFCB	**Scalar**	**Cell-by-cell budget flag and unit:** **<0 – Constant-head cell-by-cell budget data are written to the Listing File.** **0 – No cell-by-cell budget** **>0 – Unit number for saving cell-by-cell budget data**
IWDFLG	**Scalar**	**Wetting flag:** **0 – Wetting is inactive.** **not 0 – Wetting is active in layers where LAYCON is 1 or 3.**
IWETIT	**Scalar**	**The iteration interval for attempting to wet cells. Wetting is attempted every IWETIT iterations.**
IHDWET	**Scalar**	**Flag indicating which equation to use for defining the head at a cell that has just converted from dry to wet:** **0 – HNEW= BOT+WETFCT(H_n-BOT)** **not 0 – HNEW = BOT+WETFCT(THRESH)**
WETFCT	**Scalar**	**Factor included in the calculation of head at a cell that has just converted from dry to wet.**
HDRY	**Scalar**	**When a cell converts to dry, HNEW is set equal to HDRY.**
LAYCON	**NLAY**	**Layer-type code:** **0 – Confined** **1 – Unconfined** **2 – Partially convertible** **3 – Fully convertible**
LAYAVG	**NLAY**	**Interblock transmissivity flag.** **0 – Harmonic mean** **10 – Arithmetic mean** **20 – Logarithmic mean** **30 – Arithmetic-mean saturated thickness and logarithmic-mean hydraulic conductivity.**
TRPY	**NLAY**	**Ratio of transmissivity (or hydraulic conductivity) in the column direction to transmissivity (or hydraulic conductivity) in the row direction.**
HY	**NCOL,NROW,nhy**	**Hydraulic conductivity. The third dimension, nhy, is the number of layers where LAYCON is 1 or 3.**
SC1	**NCOL,NROW,NLAY**	**Primary storage capacity. Only allocated when simulation is transient.**
SC2	**NCOL,NROW,ntop**	**Secondary storage capacity. Only allocated when simulation is transient. The third dimension, ntop, is the number of layers where LAYCON is 2 or 3.**
WETDRY	**NCOL,NROW,nwet**	**Wetting threshold combined with wetting direction indicator. Absolute value is the wetting threshold. Negative indicates wetting only from below. Zero indicates no wetting. Positive indicates wetting from sides and below. The third dimension, nwet, is the number of layers where wetting can occur.**
CVWD	**NCOL,NROW,NLAY-1**	**Vertical conductance between cells. This is allocated only if wetting is active.**

BCF consists of six primary subroutines: GWF2BCF7AR, GWF2BCF7AD, GWF2BCF7FM, GWF2BCF7BDADJ, GWF2BCF7BDS, and GWF2BCF7BDCH. The budget procedure consists of three subroutines because BCF computes three budget terms: flow between adjacent cells, storage, and constant-head flow. BCF is similar to the Layer-Property Flow (LPF) Package; therefore, detailed documentation for BCF is not included. Refer to the LPF section for more details.

GWF2BCF7AR

This subroutine allocates and reads BCF data. Parameters are not supported, so GWF2BCF7AR is not as complex as the comparable LPF subroutine. However, the input data for each layer depends on the layer-type code, LAYCON. Therefore, this subroutine must test LAYCON to determine the arrays that are allocated and read for each layer. There are four main parts of this subroutine.

In the first part of the code (comments 1–5) initial records are read from the BCF input file to determine the options in effect and the resulting memory requirements. Quite a few lines of code are required to decode the layer-type and inter-block transmissivity codes and to write information to the Listing File. LAYHDT is set to 0 for when LAYCON is 0 or 2, and LAYHDT is set to 1 when LAYCON is 1 or 3. LAYHDS is set to 0 when LAYCON is 0 or 1, and LAYHDS is set to 1 when LAYCON is 2 or 3.

In the second part of the code (comment 6) the required memory is allocated. When an array is unneeded, a single element is allocated so that all pointers are associated.

In the third part of the code (comments 7–8) the arrays are read. The layer arrays are read through one do loop over the number of model layers (NLAY). All layer arrays are read using utility subroutine U2DREL.

Finally, the data are checked and preliminary computations are made by calling secondary subroutine SGWF2BCF7 (comment 9).

Arguments:

IN – Input unit number for BCF Package
IGRID – The grid number, which is used for local grid refinement

GWF2BCF7AD

This subroutine prepares BCF data for a new time step. This is done only when wetting is active. HOLD for dry cells that are eligible for conversion to wet is set equal to the bottom elevation. This is necessary to cause the change in head over the time step to be properly computed for the storage term in the flow equation. Otherwise, HOLD would have a value of HNOFLO or HDRY depending on whether the cell was initially dry or converted to dry in a prior time step.

Arguments:

KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

GWF2BCF7FM

This subroutine computes conductance and storage terms in the flow equation. The computations depend to a great extent on the layer type. For confined layers, cell drying and wetting are not allowed, and conductance has already been computed in GWF2BCF7AR. Convertible cells require computation of conductance and conversion between wet and dry, which is done by calling secondary subroutine SGWF2BCF7H.

Arguments:

KITER – Current iteration of solver
KSTP – Current time step
KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

GWF2BCF7BDS

This subroutine computes the storage budget term.

Arguments:

KSTP – Current time step
KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

GWF2BCF7BDCH

This subroutine computes the constant-head budget term.

Arguments:

KSTP – Current time step
KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

GWF2BCF7BDADJ

This subroutine computes the flow between adjacent cells, which is done only if needed. These flows are needed when saving cell-by-cell budget data to a file and when returning them in array BUFF as indicated by variable IBDRET. If values are being returned but not saved, the flows are computed only for a subgrid determined by arguments IC1, IC2, IR1, IR2, IL1, and IL2.

Arguments:

KSTP – Current time step
KPER – Current stress period
IDIR – Coordinate direction flag:
- 1 – Across columns (through right face)
- 2 – Across rows (through front face)
- 3 – Across layers (through lower face)

IBDRET – Flag for returning budget values in BUFF:
- 0 – Do not return values
- not 0 – Return values

IC1 – First column of subgrid
IC2 – Last column of subgrid
IR1 – First row of subgrid
IR2 – Last row of subgrid
IL1 – First layer of subgrid
IL2 – Last layer of subgrid
IGRID – The grid number, which is used for local grid refinement

Secondary Subroutines

SGWF2BCF7N

This secondary subroutine is called by GWF2BCF7AR to check data for consistency and to perform some initial calculations. A secondary subroutine is used because of the size of the code.

Arguments:

ISS – Steady-state flag:
  0 – At least one transient stress period
  not 0 – All stress periods are steady state.

SGWF2BCF7N performs its work in the following sequence:

1. Vertical conductance is computed by multiplying vertical leakance by cell area. GWF2BCF7AR has read vertical leakance into array CV.
2. Vertical conductance is saved in CVWD if wetting is active.
3. If IBOUND is 0, horizontal and vertical conductance (CC and CV) are set equal to 0. (A user might make CC and CV arrays constant for all cells as a matter of input convenience.)
4. A check is made to see if IBOUND is not 0 when transmissive properties in all directions are 0. Numeric problems occur in the solvers if a flow equation is constructed in which all conductances to adjacent cells are 0. Accordingly, when this situation is detected for a cell, IBOUND is set to 0 and a message is written to the Listing File. Also, head is set equal to 888.88 as an indicator that the cell has been converted to no flow.
5. For layers where transmissivity is constant, horizontal conductance is computed by calling one of several secondary subroutines as determined by the user-specified option for computing inter-block transmissivity, LAYAVG.
6. Confined storage coefficient and specific yield are multiplied by cell area to obtain storage capacity.

SGWF2BCF7H

This secondary subroutine computes horizontal conductance for a layer from saturated thickness and hydraulic conductivity. SGWF2BCF7H is called by GWF2BCF7FM for water table and fully convertible layers. SGWF2BCF7H converts dry cells to wet if wetting is active and computes saturated thickness for cells. SGWF2BCF7H then calls another secondary subroutine for computing the horizontal branch conductances according to the user-specified option for computing interblock transmissivity (LAYAVG).

Arguments:

K – Layer for which horizontal conductance is being calculated.
KB – Third index of arrays HY and WETDRY corresponding to layer K of the grid
KITER – Current iteration of solver
KSTP – Current time step
KPER – Current stress period

SGWF2BCF7C

This secondary subroutine computes horizontal conductance between nodes using harmonic mean transmissivity. SGWF2BCF7C can be called by SGWF2BCF7H and SGWF2BCF7N. Upon entry, CC contains the transmissivity for cells.

Arguments:

K – Layer for which conductance is being calculated.

SGWF2BCF7A

This secondary subroutine computes horizontal conductance using arithmetic-mean transmissivity. SGWF2BCF7A can be called by SGWF2BCF7H and SGWF2BCF7N. Upon entry, CC contains the transmissivity for cells.

Arguments:

K – Layer for which conductance is being calculated.

SGWF2BCF7L

This secondary subroutine computes horizontal conductance using logarithmic mean transmissivity. SGWF2BCF7L can be called by SGWF2BCF7H and SGWF2BCF7N. Upon entry, CC contains the transmissivity for cells.

Arguments:

K – Layer for which conductance is being calculated.

SGWF2BCF7U

This secondary subroutine computes horizontal conductance using arithmetic mean saturated thickness and logarithmic mean hydraulic conductivity. SGWF2BCF7U can be called by SGWF2BCF7H. Upon entry, CC contains the hydraulic conductivity and BUFF contains the saturated thickness for cells.

Arguments:

K – Layer for which conductance is being calculated.

Layer-Property Flow Package

The Layer-Property Flow (LPF) Package computes conductance terms for flow between cells and storage terms. When simulating confined flow, the computations are straightforward, but complexities such as unconfined conditions, cell drying, cell wetting, and conversion between confined and unconfined conditions cause the code to be much more complex. Support for parameters also adds complexity to the code. Shared data for the LPF Package are declared in Fortran Module GWFLPFMODULE and defined in table 9–6.

Table 9–6. Variables in Fortran module GWFLPFMODULE.

Variable Name	Size	Description
ILPFCB	Scalar	Cell-by-cell budget flag and unit: <0 – Constant-head cell-by-cell budget data are written to the Listing File. 0 – No cell-by-cell budget >0 – Unit number for saving cell-by-cell budget data
IWDFLG	Scalar	Wetting flag: 0 – Wetting is inactive. not 0 – Wetting is active in at least one layer.
IWETIT	Scalar	The iteration interval for attempting to wet cells. Wetting is attempted every IWETIT iteration.
IHDWET	Scalar	Flag indicating which equation to use for defining the head at a cell that has just converted from dry to wet: 0 – HNEW= BOT+WETFCT(Hn-BOT) not 0 – HNEW = BOT+WETFCT(THRESH)
ISFAC	Scalar	SFAC option flag: 0 – Not active not 0 – Active
ICONCV	Scalar	CONSTANTCV option flag: 0 – Not active not 0 – Active
ITHFLG	Scalar	THICKSTRT option flag: 0 – Not active not 0 – Active
NOCVCO	Scalar	NOCVCORRECTION option flag: 0 – Not active not 0 – Active
WETFCT	Scalar	Factor included in the calculation of head at a cell that has just converted from dry to wet.
HDRY	Scalar	When a cell converts to dry, HNEW is set equal to HDRY.
LAYTYP	NLAY	Layer-type code: 0 – A confined layer >0 – A convertible layer <0 – Convertible unless THICKSTRT option is active, in which case the layer is confined. After detecting that the layer should be confined, GWF2LPF7AR changes LAYTYP to 0.
LAYAVG	NLAY	Interblock transmissivity flag for layers: 0 – Harmonic mean 1 – Logarithmic mean 2 – Arithmetic-mean saturated thickness and logarithmic-mean hydraulic conductivity
CHANI	NLAY	Horizontal anisotropy flag or value for layers: ≤0 – Array HANI defines horizontal anisotropy for each cell in the layer. >0 – CHANI is the horizontal anisotropy for the entire layer.
LAYVKA	NLAY	Vertical anisotropy flag for layers: 0 – VKA contains vertical hydraulic conductivity. not 0 – VKA contains the ratio of horizontal to vertical hydraulic conductivity.
LAYWET	NLAY	Wetting flag for layers: 0 – Wetting is inactive. not 0 – Wetting is active.
LAYSTRT	NLAY	Flag indicating layers for which LAYTYP was negative when THICKSTRT was active: 0 – LAYTYP ≥0 or THICKSTRT is not active. not 0 – LAYTYP <0 and THICKSTRT is active.

LAYFLG	6,NLAY	Print codes for printing arrays when they are defined by parameters (k is the layer index): (1,k) – Print code for HK values (2,k) – Print code for VKA values (3,k) – Print code for SC1 values (4,k) – Print code for SC2 values (5,k) – Print code for VKCB values (6,k) – Print code for HANI values
HK	NCOL,NROW,NLAY	Horizontal hydraulic conductivity in the row direction.
VKA	NCOL,NROW,NLAY	Vertical hydraulic conductivity or the ratio of horizontal to vertical hydraulic conductivity depending on LAYVKA.
VKCB	NCOL,NLAY,ncb	Vertical hydraulic conductivity of Quasi–3D confining bed, where ncb is the number of Quasi–3D confining beds.
SC1	NCOL,NROW,NLAY	Confined storage capacity.
SC2	NCOL,NROW,ncvt	Unconfined storage capacity, where ncvt is the number of convertible layers.
HANI	NCOL,NROW,nhani	Horizontal anisotropy, where nhani is the number of layers in which horizontal anisotropy is not constant (see CHANI).
WETDRY	NCOL,NROW,nwet	Wetting threshold combined with wetting direction indicator. Absolute value is the wetting threshold. Negative indicates wetting only from below. 0 indicates no wetting. Positive indicates wetting from sides and below. The third dimension, nwet, is the number of layers where wetting can occur.

LPF consists of six primary subroutines: GWF2LPF7AR, GWF2LPF7AD, GWF2LPF7FM, GWF2LPF7BDADJ, GWF2LPF7BDS, and GWF2LPF7BDCH. The budget procedure consists of three subroutines because LPF computes three budget terms: flow between adjacent cells, storage, and constant-head flow.

GWF2LPF7AR

This subroutine allocates and reads LPF data. GWF2LPF7AR is fairly complex because it reads a large amount of data, there are many complex data dependencies, and much of the data can be optionally defined using parameters.

Arguments:

IN – The input unit number for the LPF Package
IGRID – The grid number, which is used for local grid refinement

The AR subroutine performs its work in the following sequence:

1. Allocate scalar data. This allows use with local grid refinement, in which this subroutine can be called multiple times to establish multiple grids. The result is that separate memory is allocated for each of the grids.
2. Identify package.
3. Read and write comments and item 1. Check for the SFAC, CONSTANTCV, THICKSTRT, and NOCVCORRECTION options.
4. Allocate and read indicator arrays for layers.
   - 4A. Print table of option codes for each layer. Set LAYHDT and LAYHDS to 0 for confined cells and to 1 for convertible cells. Set LAYSTRT=1 and set LAYTYP=0 if LAYTYP<0 and THICKSTRT is active.
   - 4B. Look through indicator arrays to find out how many layer arrays are needed. Print a second table showing the options for each layer in text form.
   - 4C. Print wetting information.
5. Allocate layer arrays. When an array is unneeded, a single element is allocated so that all pointers are associated.
6. Read parameter definitions. Create flags for each parameter type—the flag is 0 if no parameter of that type is specified and 1 if one or more parameters of that type are specified. If VK or VANI parameters are defined, then SGWF2LPF7CK is called to check that the layers associated with the parameter correspond to the value of LAYVKA. For example, all clusters for a VANI parameter should specify layers for which LAYVKA is not 0.
   - 6A. If any HANI parameters are defined, then horizontal anisotropy for all layers must be defined using parameters. Thus, CHANI cannot be a positive number for any layer, which indicates that horizontal anisotropy is a constant for a layer.

7. Define arrays for each layer. All arrays except for WETDRY can be defined either by directly reading them or by using parameters. WETDRY can be defined only by directly reading it. If an array is directly read, U2DREL is called to read the array. If an array is defined using parameters, UPARARRSUB1 is called to define the values. After an array is defined using parameters, UPARARRCK is called to check that a value has been defined for every cell. After SC1 and SC2 arrays are defined, secondary subroutine SGWF2LPF7SC is called to compute storage capacity.
8. Prepare and check data. This is performed by secondary subroutine SGWF2LPF7N.
9. Save grid pointers and return.

GWF2LPF7AD

This subroutine prepares LPF data for a new time step. This is done only when wetting is active and the stress period is transient. HOLD for dry cells that are eligible for conversion to wet is set equal to the bottom elevation. This is necessary to cause the change in head over the time step to be properly computed for the storage term in the flow equation. Otherwise, HOLD would have a value of HNOFLO or HDRY depending on whether the cell was initially dry or converted to dry in a prior time step.

Arguments:

KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

GWF2LPF7FM

This subroutine computes conductance and storage terms in the flow equation. The computations depend to a great extent on the layer type. For confined layers, cell drying and wetting are not allowed, and conductance has already been computed in GWF2LPF7AR. Convertible cells require computation of conductance and conversion between wet and dry.

Arguments:

KITER – Current iteration for the solver
KSTP – Current time step
KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

The FM subroutine performs its work in the following sequence:

1. Set pointers to the LPF data for the grid, get the steady-state flag for the current stress period, and define the constant ONE.
2. Loop through all layers and compute conductance for convertible layers. If a layer is convertible, call subroutine SGWF2LPF7HCOND to compute horizontal conductance. If a layer is convertible or the layer below is convertible, call SGWF2LPF7VCOND to compute vertical conductance between layers.
3. If stress period is transient, loop through all layers to compute storage terms—steps 4–5. Compute 1/DELT, which is needed to compute storage terms.
4. Test layer to see if convertible or confined.
5. Compute storage terms for a confined layer. SC1 contains storage capacity computed from specific storage. Subtract SC1/DELT from HCOF, and subtract SC1*HOLD/DELT from RHS.
6. Compute storage terms for a convertible layer.
7. Loop through all layers and apply the leakage correction if needed—steps 8 and 9.
8. Compute leakage correction to layer above if the current layer is convertible.
9. Compute leakage correction to layer below if the layer below is convertible.
10. Return.

GWF2LPF7BDADJ

This subroutine computes the flow between adjacent cells, which is done when saving cell-by-cell budget data to disk and when returning them in array BUFF as indicated by variable IBDRET.

Arguments:

KSTP – Current time step
KPER – Current stress period
IDIR – Coordinate direction flag:
 1 – Across columns (through right face)
 2 – Across rows (through front face)
 3 – Across layers (through lower face)
IBDRET – Flag for returning budget values in BUFF:
 0 – Do not return values.
 not 0 – Return values.
IC1 – First column of transport subgrid
IC2 – Last column of transport subgrid
IR1 – First row of transport subgrid
IR2 – Last row of transport subgrid
IL1 – First layer of transport subgrid
IL2 – Last layer of transport subgrid
IGRID – The grid number, which is used for local grid refinement

GWF2LPF7BDS

This subroutine computes the storage budget term.

Arguments:

KSTP – Current time step
KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

GWF2LPF7BDCH

This subroutine computes the constant-head budget term.

Arguments:

KSTP – Current time step
KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

Secondary Subroutines

SGWF2LPF7N

This secondary subroutine is called by GWF2LPF7AR to check data for consistency and perform some initial calculations. A secondary subroutine is used because of the size of the code.

Arguments: None

This subroutine performs its work in the following sequence:

1. Define constants ZERO and HCNV. HNEW will be set equal to HCNV at cells that are converted to no flow because all hydraulic conductivity values are 0.
2. Loop through all layers to check that the aquifer hydraulic conductivity in at least one direction is not 0 when IBOUND is not 0. Numerical problems occur in the solvers if a flow equation is constructed in which all conductances to adjacent cells are 0. Accordingly, when this situation is detected for a cell, IBOUND is set to 0 and a message is written to the Listing File. Also, head is set equal to 888.88 as an indicator that the cell has been converted to no flow. The checks depend on whether wetting is active.
3. Wetting is active, so IBOUND and WETDRY must be checked to see if a cell is wet or can become wet.
4. Wetting is inactive, so only IBOUND must be checked to see if a cell is wet.
5. For confined layers, compute horizontal conductance by calling secondary subroutine SGWF2LPF7HCOND.
6. For confined layers, compute vertical conductance by calling secondary subroutine SGWF2LPF7VCOND. Vertical conductance between two layers is constant only when both layers are confined.
7. Return.

SGWF2LPF7HCOND

This subroutine is called by GWF2LPF7FM and SGWF2LPF7N to compute horizontal branch conductance for a layer. SGWF2LPF7HCOND works for confined or convertible layers.

Arguments:

K – Layer for which conductance is being computed
KITER – Current iteration for the solver
KSTP – Current time step
KPER – Current stress period

SGWF2LPF7WET

This subroutine is called by SGWF2LPF7HCOND to scan all cells in a wettable layer and converts dry cells to wet according to the wetting criteria.

Arguments:

K – Layer for which dry cells are being converted.
KITER – Current iteration for the solver
KSTP – Current time step
KPER – Current stress period
IHDCNV – Cell conversion label print flag:
    0 – Label for cell conversion table has not been printed.
    not 0 – Label for cell conversion table has been printed.
NCNVRT – The number of cells in buffer arrays ICNVRT, JCNVRT, and ACNVRT
ICNVRT – Row indices for cells that convert
JCNVRT – Column indices for cells that convert
ACNVRT – Labels for cells that convert

This subroutine performs its work in the following sequence:

1. After defining the constant ZERO, loop through all cells of the layer. The layer is first argument, which is K.
2. Test to see if the cell is dry and that WETDRY is not 0. If so, the cell is eligible to convert to wet, and steps 3–7 are followed to find out if the cell should convert.
3. Compute the wetting elevation, TURNON, which is the absolute value of WETDRY plus the bottom elevation.
4. Check head in the cell below to see if the head exceeds the wetting elevation. If so, then GO TO statement 50, which converts the cell to wet.

5. If WETDRY is positive, then head in four surrounding cells also can cause conversion to wet. Check the four adjacent cells one at a time. In addition to the head exceeding the wetting threshold, the adjacent cell also must not have converted to wet this iteration. Otherwise, one cell going wet could cause an avalanche across the grid. An IBOUND value of 30000 indicates a cell that just went wet. If the wetting elevation is exceeded, jump to statement 50, which converts the cell to wet.
6. The wetting criteria have not been met, so go to the next cell by jumping to statement 100, which is the end of the loop.
7. Convert the cell to wet. Call SGWF2LPF7WDMSG to write a message saying the cell converted to wet. Select the equation to use for the head in the converted cell and set the head. Set IBOUND equal to 30000 as an indicator that the cell is wet but that it just converted to wet. This will be changed to 1 after returning to SGWF2LPF7HCOND.
8. This statement ends the loop for all cells in the layer.
9. Return.

SGWF2LPF7WDMSG

This subroutine is called by SGWF2LPF7HCOND or SGWF2LPF7WET to store and print wet and dry messages. To save space, five messages are printed on a line. NCNVRT counts the number of messages in the buffer. After five messages have accumulated or when the ICODE argument is 0 with any number of messages, a line is printed. Before printing the first line, a title is printed. IHDCNV is switched from 0 to 1 after the title is printed.

The format for printing is changed if the number of rows or columns is greater than 999. Five digits are used for row and column numbers rather than three. This makes the line a little longer so that the title does not align perfectly and packs the values close together, but the fields will not overflow.

Arguments:

ICODE – Operation code:
- 0 – Print a partially full buffer.
- not 0 – Add a cell to the buffer.

NCNVRT – The number of cells in buffer arrays ICNVRT, JCNVRT, and ACNVRT
ICNVRT – Row indices for cells that convert
JCNVRT – Column indices for cells that convert
ACNVRT – Labels for cells that convert
IHDCNV – Cell conversion label print flag:
- 0 – Label for cell conversion table has not been printed.
- not 0 – Label for cell conversion table has been printed.

IOUT – File unit number for Listing File
KITER – Current iteration for the solver
J – Column of converted cell
I – Row of converted cell
K – Layer of converted cell
KSTP – Current time step
KPER – Current stress period
NCOL – Number of columns in the grid
NROW – Number of rows in the grid

SGWF2LPF7HHARM

This secondary subroutine computes horizontal conductance between nodes using harmonic mean transmissivity. SGWF2LPF7HHARM can be called by SGWF2BCF7HCOND. Upon entry, CC contains the cell thickness.

Arguments:

K – Layer for which conductance is being computed.

SGWF2LPF7HLOG

This secondary subroutine computes horizontal conductance between nodes using logarithmic mean transmissivity. SGWF2LPF7HLOG can be called by SGWF2BCF7HCOND. Upon entry, CC contains the cell thickness.

Arguments:

K – Layer for which conductance is being computed.

SGWF2LPF7HUNCNF

This secondary subroutine computes horizontal conductance between nodes using arithmetic mean saturated thickness and logarithmic mean transmissivity. SGWF2LPF7HUNCNF can be called by SGWF2BCF7HCOND. Upon entry, CC contains the cell thickness.

Arguments:

K – Layer for which conductance is being computed.

SGWF2LPF7VCOND

This subroutine is called by GWF2LPF7FM and SGWF2LPF7N to compute vertical branch conductance between a layer and the layer below.

Arguments:

K – Upper layer for which conductance is being computed.

SGWF2LPF7SC

This subroutine is called by SGWF2LPF7AR to compute storage capacity. If the ISPST argument is not 0, argument SC is specific storage. SC is multiplied by cell area and cell thickness to get storage capacity. If ISPST is 0, SC is specific yield. SC is multiplied by cell area to get storage capacity.

Arguments:

K – Layer for which conductance is being computed.

SGWF2LPF7CK

If a VK or VANI parameter is defined, then SGWF2LPF7CK is called by GWF2LPF7AR to check that the layers associated with the parameter correspond to the value of LAYVKA. All clusters for a VANI parameter should specify layers for which LAYVKA is not 0. All clusters for a VK parameter should specify layers for which LAYVKA is 0.

Arguments:

SC – Specific storage or specific yield that will be converted to storage capacity.
K – Layer for which storage capacity is being computed.
ISPST – Flag indicating the data initially in SC:
    0 – Specific yield
    not 0 – Specific storage

Horizontal Flow Barrier Package

The Horizontal Flow Barrier (HFB) Package simulates flow barriers by reducing horizontal conductance. Shared data for the HFB Package are declared in Fortran Module GWFHFBMODULE and defined in table 9–7.

Table 9–7. Variables in Fortran module GWFHFBMODULE.

Variable Name	Size	Description
MXHFB	Scalar	The second dimension of HFB, which includes space for the active horizontal-flow barriers and the parameter definitions for horizontal-flow barriers.
NHFB	Scalar	The number of active horizontal-flow barriers.
IPRHFB	Scalar	Flag for printing HFB data – 0 indicates do not print, 1 indicates print.
NHFBNP	Scalar	The number of nonparameter horizontal-flow barriers.
NPHFB	Scalar	The number of HFB parameters.
IHFBPB	Scalar	The value of the second index in HFB at which parameter data begins.
HFB	7,MXHFB	Horizontal-flow barrier list, which includes space for the active horizontal-flow barriers and the parameter definitions for horizontal-flow barriers.

The HFB Package consists of two primary subroutines GWF2HFB7AR and GWF2WEL7FM, and four secondary subroutines SGWF2HFB7MC, SGWF2HFB7CK, SGWF2HFB7RL, and SGWF2HFB7SUB. SGWF2HFB7CK checks to insure that the two cells that define each flow barrier are adjacent. SGWF2HFB7RL is used in the AR subroutine to read lists of barriers. SGWF2HFB7SUB is used in the AR subroutine to substitute the barriers for active parameters into the list of active barriers.

Variable LAYHDT is used to determine if HFB can be used. The value for LAYHDT will be 0 for confined layers and 1 for layers for which transmissivity is head dependent. SGWF2HFB7MC is called by the Allocate and Read Procedure subroutine to compute the modified horizontal conductance caused by barriers for confined layers. When transmissivity is head dependent, the conductance of the barrier depends on the saturated thickness, which is computed in the Formulate Procedure subroutine.

GWF2HFB7AR

This subroutine allocates memory for HFB and reads all HFB input data. This subroutine must be called after the primary internal flow package (either Block-Centered Flow or Layer-Property Flow) Formulate subroutine. GWF2HFB7AR allocates memory for HFB much as is done for stress packages that use list data such as the River and Well Packages; however, the list of data is different from the data lists for the stress packages. Each line of data for the stress packages has three Integers (layer, row, and column) while HFB has five Integers: layer, row 1, column 1, row 2, and column 2. Accordingly, HFB cannot use the ULSTRD and UPARLSTSUB utility subroutines used to handle data for the stress packages. The SGWF2HFB7RL and the SGWF2HFB7SUB routines are similar to ULSTRD and UPARLSTSUB.

If variable LAYHDT is negative, then HFB will abort the simulation because an unrecognized internal flow package is in use. The value for LAYHDT will be 0 for confined layers, and SGWF2HFB7MC is called to compute the modified horizontal conductance caused by barriers for confined layers. At confined cells, the conductance does not change during the simulation. When transmissivity is head dependent, the conductance of the barrier depends on the saturated thickness, which is computed in the Formulate Procedure subroutine.

Arguments:

IN – The input unit number for the HFB Package
IGRID – The grid number, which is used for local grid refinement

GWF2HFB7FM

This subroutine modifies horizontal conductance between convertible cells for which a horizontal-flow barrier is specified. This subroutine must be called after the primary internal flow package (either Block-Centered Flow or Layer-Property Flow) Formulate subroutine. For layers for which transmissivity is head dependent (LAYHDT=1), the modified conductance caused by the barrier is computed from the saturated thickness. At confined cells (LAYHDT=1), no additional computation is made for barriers because this has already been done in GWF2HFB7AR.

Arguments:

IGRID – The grid number, which is used for local grid refinement

SGWF2HFB7MC

This subroutine modifies horizontal conductance between confined cells for which a horizontal-flow barrier is specified. This subroutine must be called after the primary internal flow package (either Block-Centered Flow or Layer-Property Flow) Allocate and Read Procedure subroutine.

Arguments: None

SGWF2HFB7CK

This subroutine checks a group of flow barriers to insure that the two cells that define each flow barrier are adjacent. This subroutine is called for each HFB parameter and for the list of barriers defined without parameters.

Arguments:

IB1 – Location in HFB of first barrier to check
IB2 – Location in HFB of last location to check

SGWF2HFB7RL

This subroutine reads a list of barriers and writes the barriers to the Listing File.

Arguments:

NLIST – Number of horizontal-flow barriers to read
HFB – List of horizontal-flow barriers
LSTBEG – Value of second index of HFB for first cell location
MXHFB – Dimensioned size of second index of HFB
INPACK – File unit number for reading barrier data
IOUT – File unit number for Listing File
LABEL – Label to be written above the list of barriers in the Listing File
NCOL – Number of columns in the grid
NROW – Number of rows in the grid
NLAY – Number of layers in the grid
IPRFLG – Print flag:
  Not 1 – Do not write barriers to Listing File.
  1 – Write barriers to Listing File.

SGWF2HFB7SUB

This subroutine reads a parameter name and substitutes values into the list of active barriers.

Arguments:

IN – File unit number for reading barrier data
PACK – Package name
IOUTU – Absolute value is the file unit number for Listing File. Negative value indicates do not print.
PTYP – Parameter type
HFB – Horizontal-flow barrier list
LSTVL – Dimensioned size of first index of HFB
MXHFB – Dimensioned size of second index of HFB
MXACTFB – Value of the second index of HFB that is the end of the active data for the current stress period
NHFB – The number of horizontal-flow barriers in the active part of HFB for the current stress period
LABEL – Label to be written above the list of barriers in the Listing File

Well Package

The Well (WEL) Package adds terms to the flow equation to represent wells. Shared data for the WEL Package are declared in Fortran Module GWFWELMODULE and defined in table 9–8.

Table 9–8. Variables in Fortran module GWFWELMODULE.

[“C*n” in size column indicates a character variable of n characters]

Variable Name	Size	Description
NWELLS	Scalar	The number of wells in the current stress period.
MXWELL	Scalar	The second dimension of WELL, which includes space for the active wells in a stress period and all wells defined by parameters.
NWELVL	Scalar	The number of the first dimension of WELL array, which includes four input values, the auxiliary data, and the well budget term.
IWELCB	Scalar	Cell-by-cell budget flag and unit. Negative indicates cell-by-cell budget is written to the Listing File, 0 indicates no cell-by-cell budget, and positive is the unit number for saving cell-by-cell budget data.
IPRWEL	Scalar	Flag for printing well data – 0 indicates do not print, 1 indicates print.
NPWEL	Scalar	The number of well parameters.
IWELPB	Scalar	The value of the second index in WELL at which parameter data begins.
NNPWEL	Scalar	The number of nonparameter wells in the current stress period.
WELAUX	C*16,20	The name of auxiliary variables.
WELL	NWELVL,MXWELL	Well list, which includes space for the active wells in a stress period and all wells defined by parameters.

The Well (WEL) Package consists of four primary subroutines: GWF2WEL7AR, GWF2WEL7RP, GWF2WEL7FM, and GWF2WEL7BD. Subroutine ULSTRD is used in the AR subroutine to read lists of wells that are defined by using parameters and in the RP subroutine to read the lists of wells that are defined without using parameters. Details of the code are not contained here. The code is similar to, but simpler than, the River Package, which is fully documented later in this chapter.

Recharge Package

The Recharge (RCH) Package adds terms to the flow equation to represent areal recharge to the ground-water system. Shared data for the RCH Package are declared in Fortran Module GWFRCHMODULE and defined in table 9–9.

Table 9–9. Variables in Fortran module GWFRCHMODULE.

Variable Name	Size	Description
NRCHOP	Scalar	The recharge option: 1 – Layer 1 2 – Layer specified in IRCH 3 – Uppermost variable-head cell
IRCHCB	Scalar	The unit number for saving RCH budget data.
NPRCH	Scalar	The number of RCH parameters.
IRCHPF	Scalar	The format code for printing recharge data defined using parameters.
RECH	NCOL,NROW	Recharge rate. Initially, recharge flux is read into RECH and then multiplied by cell area.
IRCH	NCOL,NROW	Layer receiving recharge when NRCHOP is 2 or 3.

The Recharge (RCH) Package consists of four primary subroutines: GWF2RCH7AR, GWF2RCH7RP, GWF2RCH7FM, and GWF2RCH7BD. Details of the code are provided for each subroutine.

GWF2RCH7AR

Arguments:

IN – The input unit number for the RCH Package
IGRID – The grid number, which is used for local grid refinement

The AR subroutine performs its work in the following sequence:

1. Allocate scalar variables. This is necessary for the grid refinement capability. Each invocation of this AR subroutine will cause new memory to be allocated. In step 9, a pointer to this memory location is saved.
2. Identify package and initialize the parameter format print flag.
3. Read the recharge option and the cell-by-cell budget flag. The format depends on the free format flag, IFREFM, which is defined in the Basic Package.
4. Check if NRCHOP is valid. If invalid, write a message and stop the simulation.
5. Write the recharge option.
6. If cell-by-cell budget will be written (IRCHCB >0), then write IRCHCB, which is the unit number for saving cell-by-cell budget data.
7. Allocate memory for RECH and IRCH.
8. Read named parameters, if any.
9. Save recharge pointers for the current subgrid. Return.

GWF2RCH7RP

Arguments:

IN – The input unit number for the RCH Package
IGRID – The grid number, which is used for local grid refinement

The RP subroutine performs its work in the following sequence:

1. Set the recharge pointers to point to the current subgrid.
2. Read flags that indicate whether data are being reused this stress period. INIRCH is read only if NRCHOP is 2. The format depends on the free format flag, IFREFM, which is defined in the Basic Package.
3. Test INRECH to see if recharge is being reused or read.
   3A. INRECH<0, which indicates recharge is being reused from previous stress period – either parameter or nonparameter values. Write a message and skip to step 5.
   3B. INRECH is greater than or equal to 0. Read recharge data for stress period.
      3BA. If no parameters, read RECH as an array. Then skip to step 4.
      3BB. If parameters have been input, define RECH by using parameters. Start by resetting any previously active parameters (Subroutine PRESET). INRECH is the number of parameters this stress period, make sure that INRECH>0. Read the parameter names and substitute the parameter values (Subroutine UPARARRSUB2) into RECH.
4. Multiply RECH by cell area.
5. Define IRCH if NRCHOP is 2. Test INIRCH to see if IRCH is being reused or read.
   5A. INIRCH<0, which indicates that IRCH is being reused from previous stress period.
   5B. INIRCH is greater than or equal to 0. Read IRCH as an array.
6. Return.

GWF2RCH7FM

Arguments:

IGRID – The grid number, which is used for local grid refinement

The FM subroutine performs its work in the following sequence:

1. Set the recharge pointers to point to the current subgrid.
2. Determine the recharge option (NRCHOP).
3. Recharge option 1—recharge is to the top layer.
   3A. Subtract recharge from RHS at cells where IBOUND>0.
4. Recharge option 2—recharge is to the layer specified in IRCH.
   4A. Subtract recharge from RHS at cells where IBOUND>0.
5. Recharge option 3—recharge is to the highest variable-head cell, except the recharge cannot be transmitted through a constant-head cell.
   5A. If the cell is constant head, skip to the next horizontal cell location.
   5B. Subtract recharge from RHS at cells where IBOUND>0.
6. Return.

GWF2RCH7BD

Arguments:

KSTP – Current time step
KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

The BD subroutine performs its work in the following sequence:

1. Set the recharge pointers to point to the current subgrid.
2. Clear the inflow and outflow flow rate accumulators.
3. Clear the BUFF array, which will hold the flow rate at each cell. Set IBD, which is the flag that indicates if cell-by-cell budget is saved. IBD=0 for no saving, IBD=1 for save using original form, and IBD=2 for save using compact budget.

4. Determine the recharge option (NRCHOP).
5. Recharge option 1—recharge is to top layer. Loop through all cells in a layer.
   5A. If the cell is variable head, save recharge rate in Q and QQ. QQ is double precision.
   5B. Also save the recharge rate in BUFF.
   5C. Add recharge to the inflow or outflow accumulators.
6. Recharge option 2—recharge is to the layer specified in IRCH. Loop through all cells in a layer.
   6A. Get the layer index for recharge from IRCH.
   6B. If the cell is variable head, save recharge rate in Q and QQ. QQ is double precision.
   6C. Also save the recharge rate in BUFF.
   6D. Add recharge to the inflow or outflow accumulators.
7. Recharge option 3—recharge is to the highest variable-head cell, except the recharge cannot be transmitted through a constant-head cell. Loop through all cells in a layer.
   7A. Initialize the layer index (IRCH) to 1, and loop through all cells in a vertical column.
   7B. If the cell is constant head, skip to next horizontal cell location.
   7C. If the cell is variable head, save recharge rate in Q and QQ. QQ is double precision.
   7D. Also save the recharge rate in BUFF, and save the layer number in IRCH.
   7E. Add recharge to the inflow or outflow accumulators.
8. Call budget appropriate utility module if cell-by-cell budget should be saved. UBUDSV is for the original format, and UBDSV3 is for the compact format.
9. Move total inflow and outflow rates into VBVL.
10. Accumulate inflow and outflow volumes into VBVL.
11. Move budget term name into VBNM.
12. Increment budget term counter (MSUM).
13. Return.

General-Head Boundary Package

The General-Head Boundary (GHB) Package adds terms to the flow equation to represent head-dependent boundaries. Shared data for the GHB Package are declared in Fortran Module GWGHBLMODULE and defined in table 9–10.

Table 9–10. Variables in Fortran module GWFGHBMODULE.

[“C*n” in size column indicates a character variable of n characters]

Variable Name	Size	Description
NBOUND	Scalar	The number of general-head boundaries in the current stress period.
MXBND	Scalar	The second dimension of BNDS, which includes space for the active general-head boundaries in a stress period and all general-head boundaries defined by parameters.
NGHBVL	Scalar	The number of the first dimension of BNDS array, which includes five input values, the auxiliary data, and the general-head boundary budget term.
IGHBCB	Scalar	The unit number for saving GHB budget data.
IPRGHB	Scalar	Flag for printing GHB data – 0 indicates do not print, 1 indicates print.
NPGHB	Scalar	The number of GHB parameters.
IGHBPB	Scalar	The value of the second index in BNDS at which parameter data begins.
NNPGHB	Scalar	The number of nonparameter general-head boundaries in the current stress period.
GHBAUX	C*16,20	The name of auxiliary variables.
BNDS	NGHBVL,MXBND	Boundary list, which includes space for the active general-head boundaries in a stress period and all general-head boundaries defined by parameters.

The General-Head Boundary (GHB) Package consists of four primary subroutines: GWF2GHB7AR, GWF2GHB7RP, GWF2GHB7FM, and GWF2GHB7BD. Subroutine ULSTRD is used in the AR subroutine to read lists of general-head boundaries that are defined using parameters and in the RP subroutine to read the lists of general-head boundaries that are defined without using parameters. Details of the code are not contained here. The code is similar to, but simpler than, the River Package, which is fully documented later in this chapter.

River Package

The River (RIV) Package adds terms to the flow equation to represent rivers. Shared data for the RIV Package are declared in Fortran Module GWFRIVMODULE and defined in table 9–11.

Table 9–11. Variables in Fortran module GWFRIVMODULE.

[“C*n” in size column indicates a character variable of n characters]

Variable Name	Size	Description
NRIVER	Scalar	The number of river reaches in the current stress period.
MXRIVR	Scalar	The second dimension of RIVR, which includes space for the active river reaches in a stress period and all river reaches defined by parameters.
NRIVVL	Scalar	The number of the first dimension of RIVR array, which includes six input values, the auxiliary data, and the river budget term.
IRIVCB	Scalar	The unit number for saving RIV budget data.
IPRRIV	Scalar	Flag for printing RIV data – 0 indicates do not print, 1 indicates print.
NPRIV	Scalar	The number of RIV parameters.
IRIVPB	Scalar	The value of the second index in RIVR at which parameter data begins.
NNPRIV	Scalar	The number of nonparameter river reaches in the current stress period.
RIVAUX	C*16,20	The name of auxiliary variables.
RIVR	NRIVVL,MXRIVR	River reach list, which includes space for the active river reaches in a stress period and all river reaches defined by parameters.

The River (RIV) Package consists of four primary subroutines: GWF2RIV7AR, GWF2RIV7RP, GWF2RIV7FM, and GWF2RIV7BD. Details of the code are provided for each subroutine.

GWF2RIV7AR

Arguments:

IN – The input unit number for the RIV Package
IGRID – The grid number, which is used for local grid refinement

The AR subroutine work is performed in the following sequence:
1. Allocate scalar variables. This is necessary for the local grid refinement capability. Each invocation of this AR subroutine will cause new memory to be allocated. In step 7, the pointers to this memory are saved.
2. Identify package and initialize NRIVER and NNPRIV.
3. READ items 0, 1, and 2. Item 0 consists of optional comments. URDCOM reads lines until a non-comment line is found. After URDCOM, LINE contains the first line after item 0, which is either item 1 or item 2 because item 1 is also optional. UPARLSTAL examines LINE to see if it is item 1, which specifies that parameters are being used. If parameters are being used, UPARLSTAL decodes the parameter information and reads the next line into LINE. Thus, after URDCOM and UPARLSTAL, LINE contains item 2 in text form. Item 2 is then decoded from LINE. The use of fixed or free format is supported.
4. Look at the end of LINE to find auxiliary variables and the no print option.
5. Allocate memory for the RIVR variable.
6. Read named parameters. For each parameter, call UPARLSTRP to read the parameter header record.
   6A. If a simple parameter, use ULSTRD to read the reaches.
   6B. If parameter has instances, for each instance call UINSRP to read the instance name and ULSTRD to read reaches.
7. Save river pointers for the current subgrid. Return.

GWF2RIV7RP

Arguments:

IN – The input unit number for the RIV Package
IGRID – The grid number, which is used for local grid refinement

The RP subroutine performs its work in the following sequence:

1. Set river pointers to point to the current subgrid.
2. Read ITMP and NPRIV using free or fixed format. NPRIV is the number of parameters, which is read only if river parameters have been defined. ITMP is a flag that specifies how nonparameter reaches are defined for the stress period. A negative value indicates reaches from the previous stress period should be reused. A non-negative value indicates the number of nonparameter reaches.
3. Calculate the number of auxiliary data values and the output unit. NRIVVL is the total number of values in RIVR for each reach. Three values define the reach cell, three values define reach properties, and one value is used for saving budget data. Thus, NRIVVL–7 is the number of auxiliary data values.
4. Determine NNPRIV, the number of nonparameter reaches for the stress period. If ITMP is non-negative, NNPRIV is ITMP. IF ITMP is negative, NNPRIV is unchanged from the last stress period.
5. Read new nonparameter reaches. Before reading them, make sure that the river reach list has sufficient room. Read the reaches using subroutine ULSTRD. Set the number of rivers (NRIVER) equal to NNPRIV.
6. Call subroutine PRESET to deactivate all river parameters. For each parameter, call UPARLSTSUB to read the parameter name and substitute reaches into the active part of RIVR. UPARLSTSUB updates NRIVER and checks to make sure RIVR has sufficient room for the data.
7. Print the number of river reaches in the stress period.
8. Return.

GWF2RIV7FM

Arguments:

IGRID – The grid number, which is used for local grid refinement

The FM subroutine performs its work in the following sequence:

1. Set river pointers to point to the current subgrid.
2. Return if no active river reaches for this time step.
3. Repeat steps 4 through 9 for each reach.
4. Get cell indices for reach from RIVR.
5. If cell is constant-head or no-flow, skip this reach.
6. Get reach properties from RIVR. RRBOT is double precision equivalent of RBOT.
7. Compare current value of head (HNEW) for cell to the elevation of the bottom of the riverbed (RRBOT), which determines whether seepage is constant or head dependent.
8. Head is greater than RRBOT, so seepage is head dependent – add terms to HCOF and RHS.
9. Head is less than or equal to RRBOT, so seepage is constant – add term to RHS only.
10. Return

GWF2RIV7BD

Arguments:

KSTP – Current time step
KPER – Current stress period
IGRID – The grid number, which is used for local grid refinement

The BD subroutine performs its work in the following sequence:

1. Set river pointers to point to the current subgrid.
2. Initialize cell-by-cell flow term flag (IBD) and budget accumulators (RATIN and RATOUT). IBD is 0 for no cell-by-cell budget. IBD is -1 for writing cell-by-cell budget to Listing File. IBD is 1 for writing cell-by-cell budget to a noncompact budget file. IBD is 2 for writing cell-by-cell budget to a compact budget file.
3. If writing cell-by-cell data in compact form, call UBDSV4 to write the budget header.
4. Clear BUFF, which always stores the cell-by-cell budget for internal use even if not written to a file.
5. If no reaches for this stress period, skip to step 7. River seepage does not exist for this time step.
6. Repeat steps 6A through 6L for each reach.
   - 6A. Get cell indices for reach from RIVR, and initialize reach seepage to 0.
   - 6B. If cell is no flow or constant head, skip to 6L. Reach seepage (RATE) already has been initialized to 0.
   - 6C. Get reach properties from RIVR.
   - 6D. Compare current value of head (HNEW) for cell to the elevation of the bottom of the riverbed (RRBOT), which determines whether seepage is constant or head dependent.
   - 6E. Head is greater than RRBOT, so compute head-dependent seepage.
   - 6F. Head is less than or equal to RRBOT, so compute constant seepage.
   - 6G. Write seepage to Listing File if IBD is negative.
   - 6H. Add seepage to BUFF.
   - 6I. Compare seepage to 0.
   - 6J. Negative seepage indicates outflow from aquifer, which is subtracted from the outflow accumulator, RATOUT.
   - 6K. Positive (or 0) seepage indicates inflow to aquifer, which is added to the inflow accumulator, RATIN.
   - 6L. Save seepage for reach if writing compact form of cell-by-cell budget.
7. Accumulate rates and volumes in VBVL. Move the budget term name into VBNM.
8. Increment the budget term number (MSUM).
9. Return.

Drain Package

The Drain (DRN) Package adds terms to the flow equation to represent drains. Shared data for the DRN Package are declared in Fortran Module GWFDRNMODULE and defined in table 9–12.

Table 9–12. Variables in Fortran module GWFDRNMODULE.

[“C*n” in size column indicates a character variable of n characters]

Variable Name	Size	Description
NDRAIN	Scalar	The number of drains in the current stress period.
MXDRN	Scalar	The second dimension of DRAI, which includes space for the active drains in a stress period and all drains defined by parameters.
NDRNVL	Scalar	The number of the first dimension of DRAI array, which includes five input values, the auxiliary data, and the drain budget term.
IDRNCB	Scalar	The unit number for saving DRN budget data.
IPRDRN	Scalar	Flag for printing DRN data – 0 indicates do not print, 1 indicates print.
NPDRN	Scalar	The number of DRN parameters.
IDRNPB	Scalar	The value of the second index in DRAI at which parameter data begins.
NNPDRN	Scalar	The number of nonparameter drains in the current stress period.
DRNAUX	C*16,20	The name of auxiliary variables.
DRAI	NDRNVL,MXDRN	Drain list, which includes space for the active drains in a stress period and all drains defined by parameters.

The Drain (DRN) Package consists of four primary subroutines: GWF2DRN7AR, GWF2DRN7RP, GWF2DRN7FM, and GWF2DRN7BD. Subroutine ULSTRD is used in the AR subroutine to read lists of drains that are defined by using parameters and in the RP subroutine to read the lists of drains that are defined without using parameters. Details of the code are not contained here. The code is similar to, but simpler than, the River Package, which is fully documented earlier in this chapter.

Evapotranspiration Package

The Evapotranspiration (EVT) Package adds terms to the flow equation to represent evapotranspiration from the ground-water system. Shared data for the EVT Package are declared in Fortran Module GWFEVTMODULE and defined in table 9–13.

Table 9–13. Variables in Fortran module GWFEVTMODULE.

Variable Name	Size	Description
NEVTOP	Scalar	The evapotranspiration option: 1 – Layer 1 2 – Layer specified in IEVT 3 – Uppermost variable-head cell
IEVTCB	Scalar	The unit number for saving EVT budget data.
NPEVT	Scalar	The number of EVT parameters.
IEVTPF	Scalar	The format code for printing evapotranspiration data defined using parameters.
EVTR	NCOL,NROW	Maximum evapotranspiration rate. Initially evapotranspiration flux is read into EVTR, and then multiplied by cell area.
EXDP	NCOL,NROW	Extinction depth.
SURF	NCOL,NROW	Elevation at which evapotranspiration becomes the maximum.
IEVT	NCOL,NROW	Layer receiving recharge when NEVTOP is 2 or 3.

The Evapotranspiration (EVT) Package consists of four primary subroutines: GWF2EVT7AR, GWF2EVT7RP, GWF2EVT7FM, and GWF2EVT7BD. Details of the code are not contained here because the code is similar to the Recharge Package, which is fully documented earlier in this chapter. The differences are minor. EVT adds terms to both RHS and HCOF; whereas RCH adds terms only to RHS. The EVT Package also reads more arrays than RCH reads, but the same approach for reading is used. When formulating the flow equation and computing the budget, EVT has one loop for all cells in a layer, and inside this loop a test is made to see which evapotranspiration option is being used. Conversely, RCH first tests to find which recharge option is being used, and for the selected option a loop runs for all the cells in one layer. The effect is the same for both approaches. The EVT approach results in a more compact code, but is slightly less efficient because the check for the option is done for every horizontal cell.

Strongly Implicit Procedure Package

The Strongly Implicit Procedure (SIP) Package is one of the solvers that can be used to solve the simultaneous equations resulting from the finite-difference approximation. Shared data for the SIP Package are declared in Fortran Module GWFSIPMODULE and defined in table 9–14.

Table 9–14. Variables in Fortran module GWFSIPMODULE.

Variable Name	Size	Description
NPARM	Scalar	The number of iteration parameters.
IPCALC	Scalar	Iteration parameter calculation flag: 0 – WSEED will be specified by the user. not 0 – WSEED will be calculated by the program.
IPRSIP	Scalar	Time step interval for printing table of maximum head change each iteration.
HCLOSE	Scalar	Head closure criterion for convergence.
ACCL	Scalar	Acceleration parameter.
W	NPARM	Iteration parameters.
EL	NCOL,NROW,NLAY	One of the diagonals in the upper triangular factor of [A+B].
FL	NCOL,NROW,NLAY	One of the diagonals in the upper triangular factor of [A+B].
GL	NCOL,NROW,NLAY	One of the diagonals in the upper triangular factor of [A+B].
V	NCOL,NROW,NLAY	Intermediate solution result.
HDCG	MXITER	Maximum head change for each iteration.
LRCH	3,MXITER	Layer, row, and column of the cell containing the maximum head change for each iteration.

The Strongly Implicit Procedure (SIP) Package consists of two primary subroutines, SIP7AR and SIP7AP. Except for the change to Fortran modules for memory allocation, SIP7 is identical to the SIP1 code originally documented in MODFLOW (McDonald and Harbaugh, 1988). Readers are referred to that documentation for additional details.

All data used by subroutine SIP7AP are passed as subroutine arguments. This results in faster execution on some computers as compared to passing data by using Fortran modules. Also for computational performance in SIP7AP, the arrays storing three-dimensional data for cells in the grid are accessed as one-dimensional arrays.

Preconditioned Conjugate-Gradient Package

The Preconditioned Conjugate-Gradient (PCG) Package is one of the solvers that can be used to solve the simultaneous equations resulting from the finite-difference approximation. Shared data for the PCG Package are declared in Fortran Module GWFPCGMODULE and defined in table 9–15.

Table 9–15. Variables in Fortran module GWFPCGMODULE.

Variable Name	Size	Description
ITER1	Scalar	The maximum number of inner iterations.
NPCOND	Scalar	Preconditioner flag: 1 – Incomplete Cholesky with row-sums agreement 2 – Polynomial
NBPOL	Scalar	When using polynomial preconditioning, flag for computing the upper bound of the maximum eigenvalue: 2 – A value of 2.0 is used. not 2 – The program computes the value.
IPRPCG	Scalar	Time step interval for printing table of maximum residual and head change each iteration.
MUTPCG	Scalar	Muting flag for printout: 0 – Print the convergence table. 1 – Print only the total number of iterations 2 – Do not print any convergence information. 3 – Print convergence information only if convergence fails.
NITER	Scalar	Inner iteration counter.
HCLOSEPCG	Scalar	Head closure criterion for convergence.
RCLOSE	Scalar	Residual closure criterion for convergence.
RELAX	Scalar	Relaxation parameter.
DAMP	Scalar	Damping parameter.
VPCG	NCOL,NROW,NLAY	Intermediate solution result. (Double precision)
SS	NCOL,NROW,NLAY	Matrix in PCG algorithm. (Double precision)
P	NCOL,NROW,NLAY	Matrix in PCG algorithm. (Double precision)
RES	NCOL,NROW,NLAY	The flow equation residual. (Double precision)
HPCG	NCOL,NROW,NLAY	Head at the beginning of an outer iteration. (Double precision)
CD	NCOL,NROW,NLAY	The main diagonal of [U] for incomplete Cholesky preconditioning.
HCSV	NCOL,NROW,NLAY	HCOF is saved in HCSV when using polynomial preconditioning.
LHCH	3,MXITER*ITER1	Layer, row, and column of the cell containing the maximum head change for each iteration.
HDCG	MXITER	Maximum head change for each iteration.
LRCHPCG	3,MXITER*ITER1	Layer, row, and column of the cell containing the maximum residual for each iteration.
IT1	MXITER*ITER1	Outer iteration flag: 0 – Not the first inner iteration of an outer iteration 1 – The first inner iteration of an outer iteration

The Preconditioned Conjugate-Gradient (PCG) Package consists of two primary subroutines, PCG7AR and PCG7AP. Except for the change to Fortran modules for memory allocation, PCG7 is the same as the PCG2 code in MODFLOW–2000. Readers are referred to Hill (1990) for additional details.

All data used by subroutine PCG7AP are passed as subroutine arguments. This results in faster execution on some computers as compared to passing data by using Fortran modules. Also for computational performance in PCG7AP, the arrays storing three-dimensional data for cells in the grid are accessed as one-dimensional arrays.

Direct Solver Package

The Direct Solver (DE4) Package is one of the solvers that can be used to solve the simultaneous equations resulting from the finite-difference approximation. Shared data for the DE4 Package are declared in Fortran Module GWFDE4MODULE and defined in table 9–16.

Table 9–16. Variables in Fortran module GWFDE4MODULE.

Variable Name	Size	Description
MXUP	Scalar	The maximum number of equations in the upper part of [A].
MXLOW	Scalar	The maximum number of equations in the lower part of [A].
MXEQ	Scalar	MXUP+MXLOW
MXBW	Scalar	The maximum value of the band width plus one.
ITMX	Scalar	The maximum number of iterations (internal or external) in one time step,
ID4DIR	Scalar	Flag that indicates the relative size of NCOL, NROW, and NLAY: 1 – NLAY≤NROW≤NCOL 2 – NLAY≤NCOL<NROW 3 – NROW<NLAY≤NCOL 4 – NROW≤NCOL<NLAY 5 – NCOL<NLAY≤NROW 6 – NCOL<NROW<NLAY
NITERDE4	Scalar	The maximum number of internal iterations in one time step.
IFREQ	Scalar	Flag indicating the frequency at which [A] changes: 0 – [A] has not changed since the previous solution. 1 – [A] changes only when the time step changes. 2 – [A] can change at the start of each stress period and when the time step size changes. 3 – [A] can change each time DE45AP is called (nonlinear flow equation).
IPRD4	Scalar	Time step interval for printing table of maximum head change for each iteration.
MUTD4	Scalar	Muting flag for printout: 0 – Print the convergence table 1 – Print only the total number of iterations 2 – Do not print any convergence information
ID4DIM	Scalar	First dimension of AU and IUPPNT: 5 – for a two-dimensional grid 7 – for a three-dimensional grid
ACCLDE4	Scalar	Acceleration parameter.
HCLOSEPCG	Scalar	Head closure criterion for convergence.
IUPPNT	ID4DIM,MXUP	The number of off-diagonal coefficients and the equation numbers of the off-diagonal coefficients that are stored in AU.
IEQPNT	NCOL,NROW,NLAY	D4 order number for model cells.
AU	ID4DIM,MXUP	The upper part of [A].
AL	MXBW,MXLOW	The matrix AL in the D4 algorithm.
D4B	MXEQ	The vector {b}.
HDCGDE4	MXITER	Maximum head change for each iteration.
LRCHDE4	3,ITMX	Layer, row, and column of the cell containing the maximum residual for each iteration.

The Direct Solver (DE4) Package consists of two primary subroutines, DE47AR and DE47AP. Except for the change to Fortran modules for memory allocation, DE4 is the same as the DE4 code in MODFLOW–2000. Readers are referred to Harbaugh (1995) for additional details.

All data used by subroutine DE47AP are passed as subroutine arguments. This results in faster execution on some computers as compared to passing data using Fortran modules.

Utility Subroutines

NonParameter Subroutines

The nonparameter utility subroutines use no data from Fortran modules. All data are passed as subroutine arguments, which are documented below. The code contains extensive comments, which provide the primary documentation. Some additional information is provided below for some of the subroutines.

URWORD

Arguments:

LINE – A line of text
ICOL – Element in LINE where the search for a word should start.
ISTART – Element in LINE that is the last character of the word.
ISTOP – Element in LINE that is the last character of the word.
NCODE – A code for converting the word:
  0 – No conversion
  1 – Convert to uppercase
  2 – Convert to Integer
  3 – Convert to Real number
N – Integer value to which the word is converted if indicated by NCODE.
R – Real number value to which the word is converted if indicated by NCODE.
IOUT – File unit number for the Listing File
IN – File unit from which the line was read.

URWORD returns the location of a word in a line of text. A word delimiter is one or more spaces, a comma, or a tab. If the word starts with a single quote, then the quote is removed from the word and the terminating delimiter must be a quote. After finding a word, the line index points to the remainder of the line so that URWORD can be called repeatedly to obtain successive words. The word can be optionally converted to uppercase, or it can be converted to an Integer or Real number. When converting to a number, the word must be 30 characters or less.

UPCASE

Arguments:

WORD – The text string to be converted

UPCASE converts a string to uppercase. The single argument, WORD, is a character variable that contains the string to be converted. The result overwrites WORD.

The algorithm works for a coding method for which the offset between a lowercase and uppercase character is the same for all alphabetic characters. The conversion is done by adding the offset between “a” and “A” to convert the character to uppercase. The conversion algorithm also relies on an assumption that the lowercase characters can be detected by using a relational test for greater than or equal to ‘a’ and less than or equal to‘z’.

URDCOM

Arguments:

IN – File unit number for reading data
IOUT – File unit number for the Listing File
LINE – A buffer for a line of text

URDCOM reads lines from file unit IN until a non-comment line is found. A comment line starts with "#". The comments are written to file unit IOUT if IOUT is greater than 0. The non-comment line that follows any comments is returned to the calling program in LINE.

ULSTRD

Arguments:

NLIST – The number of cells read in the list
RLIST – The array that holds the list of data (LDIM,MXLIST)
LSTBEG – The location in RLIST (second index) where the first data should be read.
LDIM – The first dimension of RLIST
MXLIST – The second dimension of RLIST
IAL – A flag indicating whether LDIM contains an extra location for saving budget data.
INPACK – The input unit number for the package that is calling ULSTRD
IOUT – The output unit number for the Listing File
LABEL – The text label that is written at the start of the list when writing the list to the Listing File.
CAUX – The names of auxiliary variables in the list
NCAUX – The dimension of CAUX
NAUX – The actual number of auxiliary variables used.
IFREFM – Free format flag defined by the Basic Package.
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
NLAY – The number of layers in the grid.
ISCLOC1 – The location in RLIST (first index) that is the first value in RLIST that gets scaled if a scale factor is defined for the list.
ISCLOC2 – The location in RLIST (first index) that is the last value in RLIST that gets scaled if a scale factor is defined for the list.
IPRFLG – Printout flag indicating to print the list if 1.

ULSTRD reads and writes to the Listing File a list of cell locations with data values as used to define river reaches, drains, general-head boundaries, constant-head boundaries, and wells. A lengthy list of arguments is required because of a variety of options that are supported.

ULSTLB

Arguments:

IOUT – File unit number for the Listing File
LABEL – Text to be printed in the label
CAUX – Names of auxiliary data fields
NCAUX – The dimensioned size of CAUX
NAUX – The number of auxiliary fields being used.

This subroutine prints a label for a list of data. The label consists of the text in argument LABEL plus names of auxiliary data fields.

U1DREL

Arguments:

A – The array to read
ANAME – The name of the array
JJ – The dimensioned size of the array
IN – File unit number for reading data
IOUT – File unit number for the Listing File

U1DREL reads a one-dimensional Real array using a control record to define how the data are read. This supports the original control records using numeric codes and the newer control records using words to specify options.

U2DINT

Arguments:

IA – The array to read
ANAME – The name of the array
II – The dimension size for the second index of IA
JJ – The dimensioned size for the first index of IA
K – Layer code for the array:
  <0 – Array is a cross section
  0 – No layer
  >0 – Layer number
IN – File unit number for reading data
IOUT – File unit number for the listing File

U2DINT reads a two-dimensional Integer array using a control record to define how the data are read. This supports the original control records using numeric codes and the newer control records using words to specify options.

U2DREL

Arguments:

A – The array to read
ANAME – The name of the array
II – The dimension size for the second index of A
JJ – The dimensioned size for the first index of A
K – Layer code for the array:
  <0 – Array is a cross section.
  0 – No layer
  >0 – Layer number
IN – File unit number for reading data
IOUT – File unit number for the Listing File

U2DREL reads a two-dimensional Real array using a control record to define how the data are read. This supports the original control records using numeric codes and the newer control records using words to specify options.

UCOLNO

Arguments:

NLBL1 – The start column label (number)
NLBL2 – The stop column label (number)
NSPACE – The number of blank spaces to leave at start of line
NCPL – The number of column numbers per line
NDIG – The number of characters in each column field
IOUT – The file unit number for writing the data – usually the Listing File

UCOLNO writes a column-number header, which is used before writing a two-dimensional array.

ULAPRS

Arguments:

BUF – The array to be written
TEXT – The name of the array being written
KSTP – The current time step
KPER – The current stress period
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
ILAY – The layer of the array to be written
IPRN – Format code
IOUT – The file unit number for writing the data – usually the Listing File

ULAPRS writes a two-dimensional array using strip format.

ULAPRW

Arguments:

BUF – The array to be written
TEXT – The name of the array being written
KSTP – The current time step
KPER – The current stress period
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
ILAY – The layer of the array to be written
IPRN – Format code
IOUT – The file unit number for writing the data – usually the Listing File

ULAPRS writes a two-dimensional array using wrap format.

ULAPRWC

ULAPRWC writes a two-dimensional Real array, but this subroutine first checks to see if all values are the same. If all values are the same, then the single value is printed rather than writing all of the individual values. If all values are not the same, then ULAPRW is called to print the values.

ULASAV

Arguments:

BUF – The array to be written
TEXT – The name of the array being written
KSTP – The current time step
KPER – The current stress period
PERTIM – The accumulated length of the current stress period
TOTIM – The accumulated length of the simulation
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
ILAY – The layer of the array to be written
ICHN – The file unit number for writing the data

ULASAV writes a header record and a layer of Real data to an unformatted file.

ULASV2

Arguments:

BUFF – The array to be written
TEXT – The name of the array being written
KSTP – The current time step
KPER – The current stress period
PERTIM – The accumulated length of the current stress period
TOTIM – The accumulated length of the simulation
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
ILAY – The layer of the array to be written
ICHN – The file unit number for writing the data
FMTOUT – The format for writing the data
LBLSAV – Flag for writing a header
  0 – Do not write header.
  not 0 – Write header.
IBOUND – The boundary array

ULASV2 writes a layer of Real data to a formatted file. Optionally, a header record can be written. IBOUND is passed as an argument but is unused. IBOUND is included for convenience for someone to provide a more complex replacement for ULASV2 that makes use of IBOUND to control output.

ULASV3

Arguments:

IDATA – The array to be written
TEXT – The name of the array being written
KSTP – The current time step
KPER – The current stress period
PERTIM – The accumulated length of the current stress period
TOTIM – The accumulated length of the simulation
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
ILAY – The layer of the array to be written
ICHN – The file unit number for writing the data
FMTOUT – The format for writing the data
LBLSAV – Flag for writing a header:
  0 – Do not write header.
  not 0 – Write header.

ULASV3 writes a layer of Integer data to a formatted file. A header record can be optionally written.

UBUDSV

Arguments:

KSTP – The current time step
KPER – The current stress period
TEXT – The name of the budget term
IBDCHN – The file unit number for writing budget data
BUFF – The array of data values to be written
NCOL – The number of columns in the grid
NROW – The number of rows in the grid

NLAY – The number of layers in the grid
IOUT – File unit number for the Listing File

UBUDSV writes a three-dimensional array of values to an unformatted file. This is used for writing budget data when COMPACT BUDGET is NOT specified in Output Control.

UBDSV1

Arguments:

KSTP – The current time step
KPER – The current stress period
TEXT – The name of the budget term
IBDCHN – The file unit number for writing budget data
BUFF – The array of data values to be written
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
NLAY – The number of layers in the grid
IOUT – File unit number for the Listing File
DELT – The length of the current time step
PERTIM – The accumulated length of the current stress period
TOTIM – The accumulated length of the simulation
IBOUND – The boundary array

UBDSV1 writes a three-dimensional array of values to an unformatted file. This is used for writing a full 3–D array of budget data when COMPACT BUDGET is specified in Output Control. This is no more compact than using UBUDSV, but UBDSV1 adds an extra record containing a compact budget code, DELT, PERTIM, and TOTIM. The compact budget code is 1. IBOUND is passed as an argument but is unused. IBOUND is included for convenience for someone to provide a more complex replacement for UBDSV1 that makes use of IBOUND to control output.

UBDSV2

Arguments:

KSTP – The current time step
KPER – The current stress period
TEXT – The name of the budget term
IBDCHN – The file unit number for writing budget data
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
NLAY – The number of layers in the grid
NLIST – The number of cells in the list
IOUT – File unit number for the Listing File
DELT – The length of the current time step
PERTIM – The accumulated length of the current stress period
TOTIM – The accumulated length of the simulation
IBOUND – The boundary array

UBDSV2 writes a header for a list of values to an unformatted file. This is used along with UBDSVA for writing constant-head budget data when COMPACT BUDGET is specified in Output Control. UBDSV2 writes a record containing a compact budget code, DELT, PERTIM, and TOTIM. The compact budget code is 2. UBDSV2 also writes a record containing the number of values to be written by UBDSVA. IBOUND is passed as an argument but is unused. IBOUND is included for convenience for someone to provide a more complex replacement for UBDSV2 that makes use of IBOUND to control output.

UBDSVA

Arguments:

IBDCHN – The file unit number for writing budget data
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
J – The column of the cell whose budget value is being written
I – The row of the cell whose budget value is being written
K – The layer of the cell for which a budget value is being written
Q – The budget value to be written
IBOUND – The boundary array
NLAY – The number of layers in the grid

UBDSVA writes a list value, and is called once for each list value specified by the header written by UBDSV2. This is used for writing constant-head budget data when COMPACT BUDGET is specified in Output Control. A one-dimensional cell index (based on column, row, and layer hierarchy) is written along with the data value. IBOUND and NLAY are passed as arguments but are unused. IBOUND and NLAY are included for convenience for someone to provide a more complex replacement for UBDSVA that makes use of these variables.

UBDSV3

Arguments:

KSTP – The current time step
KPER – The current stress period
TEXT – The name of the budget term
IBDCHN – The file unit number for writing budget data
BUFF – The array of data values to be written
IBUFF – An array of layer numbers for the data values
NOPT – The writing option:
  Not 1 – Values in BUFF apply to layer 1, so IBUFF is not written.
  1 – Values in BUFF apply to layers in IBUFF, so IBUFF is written.
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
NLAY – The number of layers in the grid
IOUT – File unit number for the Listing File
DELT – The length of the current time step
PERTIM – The accumulated length of the current stress period
TOTIM – The accumulated length of the simulation
IBOUND – The boundary array

UBDSV3 writes a two-dimensional array of values to an unformatted file and an optional two-dimensional array of layer numbers. This is used for writing Recharge and Evapotranspiration budget data when COMPACT BUDGET is specified in Output Control. UBDSV3 writes a record containing a compact budget code, DELT, PERTIM, and TOTIM. The compact budget code is 3 if no indicator array is written and 4 if an indicator array is written. IBOUND is passed as an argument but is unused. IBOUND is included for convenience for someone to provide a more complex replacement for UBDSV3 that makes use of IBOUND to control output.

UBDSV4

Arguments:

KSTP – The current time step
KPER – The current stress period
TEXT – The name of the budget term

NAUX – The number of auxiliary data values being used
AUXTXT – Names of auxiliary values
IBDCHN – The file unit number for writing budget data
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
NLAY – The number of layers in the grid
NLIST – The number of cells in the list
IOUT – File unit number for the Listing File
DELT – The length of the current time step
PERTIM – The accumulated length of the current stress period
TOTIM – The accumulated length of the simulation
IBOUND – The boundary array

UBDSV4 writes a header for a list of values to an unformatted file. This is used along with UBDSVB for writing budget data for the River, General-head Boundary, Well, and Drain Packages when the COMPACT BUDGET is specified in Output Control. UBDSV4 writes a record specifying the number of data values to be written for each entry in the list and a record containing a compact budget code, DELT, PERTIM, and TOTIM. The compact budget code is 5. UBDSV4 also writes a record containing the number of entries to be written by UBDSVB. IBOUND is passed as an argument, but is unused. IBOUND is included for convenience for someone to provide a more complex replacement for UBDSV4 that makes use of IBOUND to control output.

UBDSVB

Arguments:

IBDCHN – The file unit number for writing budget data
NCOL – The number of columns in the grid
NROW – The number of rows in the grid
J – The column of the cell for which a budget value is being written
I – The row of the cell for which a budget value is being written
K – The layer of the cell for which a budget value is being written
Q – The budget value to be written
VAL – The list data for the cell for which a budget value is being written
NVL – The size of VAL
NAUX – The number of auxiliary data values being used
LAUX – The location in VAL of the first auxiliary data value
IBOUND – The boundary array
NLAY – The number of layers in the grid

UBDSVB writes a list entry, which can consist of multiple data values, and is called once for each list entry specified by the header written by UBDSV4. This is used for writing budget data combined with auxiliary data for the River, General-head Boundary, Well, and Drain Packages when COMPACT BUDGET is specified in Output Control. A one-dimensional cell index (based on column, row, and layer hierarchy) is written along with the data values. IBOUND and NLAY are passed as arguments but are unused. IBOUND and NLAY are included for convenience for someone to provide a more complex replacement for UBDSVB that makes use of these variables.

UMESPR

Arguments:

TEXT1 – First part of text message
TEXT2 – Second part of text message
IOUT – File unit number for the Listing File

UMESPR writes a message to the Listing File.

USTOP

Arguments:

STOPMESS – Text message to write prior to stopping.

USTOP optionally writes a message and stops execution. Throughout MODFLOW, USTOP is called to terminate the program when an error occurs rather than directly invoking STOP. This is done to facilitate the possibility of enhancing the program to capture control upon an error.

Parameter Subroutines

The parameter utility subroutines are used to read parameter definitions and apply the definitions to generate data used to construct the flow equation. Parameters are a user-selected alternative to direct input of data into program variables. The parameters are used to generate the same model variables into which data would otherwise be read directly; therefore, the Formulate Procedure subroutines receive data in the same form whether or not parameters are used.

The variables for storing parameter definitions are defined in Fortran module PARAMMODULE, which is documented in the Basic Package section of this chapter. Parameters have 10–character names stored in PARNAM. Instance names, which designate multiple versions of the same parameter that can be applied at different times, also have 10 characters and are stored in INAME. Parameter and instance names are case insensitive except that case is maintained for purposes of printing the names in the output file. For example, the parameter name "HyCond" would not be differentiated from the parameter name "HYCOND" because they are identical without consideration of case; however, the output file would contain the form of the name that was specified in the input data. This approach is implemented by storing the names exactly as read from the input file and temporarily converting the names to uppercase whenever the list of names is searched.

PARNAM contains all parameters, regardless of the parameter type. The location in the list establishes the parameter number, which is used extensively to reference the various variables that store parameter information. The parameter type is a four-character value stored in PARTYP using the same order as in PARNAM. Similarly, B contains parameter values.

The parameter index, IPLOC, stores four indices for each parameter that are used to define the model data values from parameters. The indices have different uses depending on whether the parameter is an array or list. Arrays are considered first. Each array parameter is specified by array clusters. A cluster defines a group of cells in one layer. Clusters are stored in IPCLST. A parameter can have any number of array clusters, so two values in IPLOC are used to specify the clusters for each parameter. IPLOC(1,p) specifies the first cluster for parameter p, and IPLOC(2,p) specifies the last cluster. List parameters are defined by specifying beginning and ending locations in the list array for the corresponding parameter type. For example, a river parameter is defined by a series of locations in the RIVR array. IPLOC(1,p) is the beginning location and IPLOC(2,p) is the ending location for parameter p.

Each array cluster has 14 values. The first value is the layer number, the second value is the multiplier array number, and the third is the zone array number. Multiplier array names are 10–character values stored in MLTNAM, and zone array names are 10–character values stored in ZONNAM. A multiplier array is a layer array that is multiplied by the parameter value when generating an array using parameters. A zone array is a layer array of integer zone numbers. Zone numbers restrict the cells of a layer to which a parameter applies. Each parameter cluster contains a list of zone numbers to which the parameter applies. The zone numbers are contained in elements 5–14 of the array cluster. The fourth value of the cluster specifies the last element in the cluster that is being used. Consider cluster n, for example. If IPCLST(4,n) is 7, then there are three zone numbers, which are IPCLST(5,n), IPCLST(6,n), and IPCLST(7,n).

IPLOC(3,p) and IPLOC(4,p) are used to specify instances. These can only apply to parameters that can vary with time. IPLOC(3,p) is the number of instances, which is 0 if instances are not being used for the parameter. IPLOC(4,p) is the element of INAME that contains the name of the first instance for the parameter.

IACTIVE contains a flag for each parameter. IACTIVE indicates whether a parameter is active in the current time step. This is used for parameters that can change with time, such as recharge parameters. An inactive parameter has a 0 value. Negative 1 indicates the parameter is active all the time; for example, LPF parameters will have -1 for IACTIVE. For active time varying parameters, IACTIVE is the number of the instance that is active.

The code for many of these subroutines is tedious because several levels of indexing are involved. First a parameter number must be determined from the name or type. The parameter index, IPLOC, must be dealt with to find where data are stored. For array parameters, the clusters must be evaluated to find the appropriate multiplier and zone arrays. Instances add another level of complexity. The expectation of the author is that these subroutines will not require frequent updating. If improvements are needed, new subroutines likely will be written.

The parameter utility subroutines use the data in PARAMMODULE as needed. All other data are passed as subroutine arguments.

UPARARRAL

Arguments:

IN – Package file unit from which parameter data will be read
IOUT – File unit number for the Listing File
LINE – Character variable containing a line of text already read from IN
NP – The number of parameters that the package will use

This subroutine determines if a package will use array parameters. When parameters were added as an option to existing MODFLOW packages, a new optional item was added to the input files. This subroutine looks for that item. Upon entry to this subroutine, a line from the package input file containing the next input item is contained in variable LINE. URWORD is called to look for the word “PARAMETER,” and if found, URWORD is called again to get the number of parameters, which is variable NP. The next line is read from the input file so that LINE will contain the next input item upon returning. If “PARAMETER” is not found in LINE, NP is set to 0. A message is printed to IOUT giving the number of parameters.

UPARARRRP

Arguments:

IN – Package file unit from which parameter data will be read
IOUT – File unit number for the Listing File
NP – The parameter number of the new parameter
ILFLG – Layer flag:
    0 – Parameter defines a two-dimensional array.
    not 0 – Parameter defines a three-dimensional array.
PTYP – The parameter type of the new parameter
ITERP – The number of times the ground-water flow simulation has been run
ITVP – Time varying parameter flag:
    0 – Parameter cannot vary with time.
    not 0 – Parameter can vary with time using instances.
IACT – Value to save in IACTIVE array.

This subroutine reads data for defining one array parameter.

ITERP is an argument that specifies the number of times the ground-water flow simulation has been run. This was used in MODFLOW–2000 when doing parameter estimation. The ground-water flow simulation is repeatedly rerun with different parameter values. When ITERP is greater than 1, all the parameters will already be defined as a result of the first run of the simulation. In this case, the parameter data are read but the parameter information does not require saving and is not printed. Parameter estimation is not initially supported in MODFLOW–2005, so ITERP will always be 1.

Allowance is made for predefining parameter values as done in the MODFLOW–2000 Sensitivity file. In this case, a parameter is partly defined prior to being read by UPARARRRP. This situation is detected by finding the parameter name in PARNAM with an unspecified (blank) type. Parameter processing then continues to complete the definition. The parameter value in the Sensitivity file supersedes the value read by UPARARRRP.

UPARARRSUB1

Arguments:

ZZ – Two-dimensional array into which data are substituted
NCOL – Number of columns in ZZ
NROW – Number of rows in ZZ
ILAY – Layer number for ZZ—ILAY is 0 if ZZ is a two-dimensional array.
PTYP – Parameter type
IOUT – File unit number for the Listing File
ANAME – The name of ZZ
IPF – Print format code:
  <0 – Do not print the array after substitution.
  ≤0 – Print the array after substitution using IPF for the format code.

This subroutine substitutes all array parameters of a specified type into a two-dimensional array. This is the functional equivalent for U2DREL. This is used by the Layer Property Flow Package. For three-dimensional arrays, UPARARRSUB1 must be called once for each layer.

UPARARRSUB2

Arguments:

ZZ – Two-dimensional array into which data are substituted
NCOL – Number of columns in ZZ
NROW – Number of rows in ZZ
ILAY – Layer number for ZZ—ILAY is 0 if ZZ is a 2–D array.
NP – The number of parameters to read
IN – Package input file unit
IOUT – File unit number for the Listing File
PTYP – Parameter type
ANAME – The name of ZZ
PACK – The package name, which is printed in error messages
IPF – Print format code:
  <0 – Do not print the array after substitution.
  ≤0 – Print the array after substitution using IPF for the format code.

This subroutine reads a list of parameter names and substitutes them into a two-dimensional array. UPARARRSUB2 is similar to UPARARRSUB1. The difference is that UPARARRSUB2 applies only a limited set of parameters while UPARARRSUB1 applies all parameters of a specified type. Also, this subroutine must deal with possibility of parameter instances. The set of parameters to apply is read from an input file. This is used by the Recharge and Evapotranspiration Packages

USUB2D

Arguments:

ZZ – Two-dimensional array into which data are substituted
NCOL – Number of columns in ZZ
NROW – Number of rows in ZZ
IP – Parameter number of parameter to be substituted
ILAY – Layer number for ZZ—ILAY is 0 if ZZ is a two-dimensional array.
INIT – Initialization flag: 0 – Do not initialize.,not 0 – Initialize to zero before substituting.
NSUB – The number of values in ZZ that are substituted

This subroutine is called by UPARARRSUB1 and UPARARRSUB2 to substitute one parameter into a two-dimensional array. This adds to the values already in the array, therefore supporting the use of additive parameters. When substituting the first parameter of a type, the array must be initialized to 0, and the INIT argument specifies whether or not to initialize the array.

UPARLSTAL

Arguments:

IN – Package file unit from which parameter data will be read
IOUT – File unit number for the Listing File
LINE – Character variable containing a line of text already read from IN
NP – The number of parameters that the package will use
MXL – The number of cell locations for all parameters

This subroutine determines if a package will use list parameters. When parameters were added as an option to existing MODFLOW packages, a new optional item was added to the input files. This subroutine looks for that item. Upon entry to this subroutine, a line from the package input file containing the next input item is contained in variable LINE. URWORD is called to look for the word “PARAMETER,” and if found, URWORD is called again to get the number of parameters, which is variable NP. In addition, URWORD is called to find MXL, the total number of list entries required to define these parameters. The next line is read from the input file so that LINE will contain the next input item upon returning. If “PARAMETER” is not found, NP and MXL are set to 0. A message is printed to IOUT giving the number of parameters.

UPARLSTRP

Arguments:

LSTSUM – Count of the number of cell locations in the list array
MXLST – The maximum number of cell locations in the list array
IN – Package file unit from which parameter data will be read
IOUT – File unit number for the Listing File
NP – The parameter number for the new parameter
PACK – The package name, which is used when printing an error message
PTYPX – The parameter type that the new parameter should have
ITERP – The number of times the ground-water flow simulation has been run
NUMINST – The number of instances that the new parameter has

This subroutine reads data for defining one list parameter.

ITERP is an argument that specifies the number of times the ground-water flow simulation has been run. This was used in MODFLOW–2000 when doing parameter estimation. The ground-water flow simulation is repeatedly rerun with different parameter values. When ITERP is greater than 1, all the parameters will already be defined as a result of the first run of the simulation. In this case, the parameter data are read but the parameter information does not require saving and is not printed. Parameter estimation is not initially supported in MODFLOW–2005, so ITERP will always be 1.

Allowance is made for predefining parameter values as done in the MODFLOW–2000 Sensitivity file. In this case, a parameter is partly defined prior to being read by UPARLSTRP. This situation is detected by finding the parameter name in PARNAM with an unspecified (blank) type. Parameter processing then continues to complete the definition. The parameter value in the Sensitivity file supersedes the value read by UPARLSTRP.

The list of cell data for the parameter is not read by this subroutine. The package Allocate and Read Procedure reads this data. Typically ULSTRD is called to read the list.

UINSRP

Arguments:

I – Instance number
IN – Package file unit from which parameter data will be read
IOUT – File unit number for the Listing File
IP – Parameter number
ITERP – The number of times the ground-water flow simulation has been run

This subroutine reads and stores one instance name. This subroutine is called by UPARARRRP and all packages that read list parameters. ITERP is used as described for UPARLSTRP to determine whether a ground-water flow simulation is being run multiple times. After the first run, instance names are read for all runs, but the names are printed only for the first run.

UPARLSTSUB

Arguments:

IN – Package file unit from which parameter data will be read
PACK – Package name
IOUTU – The absolute value is the file unit number for the Listing File. A negative sign indicates that the list should not be printed.
PTYP – Parameter type
RLIST – The package list array that stores the list of data that are active in the current stress period and parameter lists
LSTVL – The first dimension of RLIST
LSTDIM – The second dimension of RLIST
NREAD – The number of values in RLIST to be moved for each cell
MXLST – The number of cells in RLIST that can be active during a stress period
NTOT – The number of cells in RLIST that are currently active
IPVL1 – The value of the first index of RLIST where substitution begins
IPVL2 – The value of the first index of RLIST where substitution ends
LABEL – Label to print above the list
CAUX – Auxiliary field names
NCAUX – The dimension of CAUX
NAUX – The number of auxiliary values being used

This subroutine reads the name of a list parameter and substitutes the parameter values into the active part of the package list, RLIST.

PRESET

Arguments:

PTYP – Parameter type.

This subroutine sets IACTIVE to 0 for all parameters of the specified type, PTYP, which makes the parameters inactive. This is called at the beginning of a stress period to deactivate the time varying parameters used in the prior stress period.

UPARLSTLOC

Arguments:

IN – Package file unit from which parameter data will be read
PACK – Package name
IOUT – The file unit number for the Listing File
PTYP – Parameter type

IBEG – The location in the package data list of the first value associated with the parameter
IEND – The location in the package data list of the last value associated with the parameter
PV – The parameter value

This subroutine reads the name of a list parameter and finds the start and end of the list of values for the parameter within the package list. This is the same as the first part of UPARLSTSUB, but rather than copying the list into the active area for the current stress period, UPARLSTLOC simply returns the indices.

UPARARRCK

Arguments:

BUFF – Temporary layer array used to count how many parameters define a value for each cell
IBOUND – Boundary array
IOUT – The file unit number for the Listing File
LAY – The layer being checked
NCOL – The number of columns in the grid
NLAY – The number of layers in the grid
NROW – The number of rows in the grid
PTYP – Parameter type

This subroutine checks to see if all the array parameters of a specified type define a value for all cells in a layer. UPARARRCK also counts the number of parameters that contribute a value to each cell. This subroutine works by going through each cluster for each parameter of the specified type as done by UPARARRSUB1. However, rather than adding to a data value at each indicated cell, it counts the times each cell is indicated. The count is stored in the temporary array, BUFF. After scanning all the appropriate clusters, BUFF is examined to see if there are any variable-head or constant-head cells where BUFF is 0. If any are found, they are printed and the simulation is aborted.

UPARFIND

Arguments:

PNAME – Name of parameter to be found
PTYP – Parameter type for the parameter to be found
CPACK – Package name to whic
IFOUND – Parameter number of the found parameter
IOUT – The file unit number for the Listing File

This subroutine looks for a parameter in the list of parameters. If found, the parameter number is returned. The parameter type must match the specified type. If the parameter is not found or the parameter type is incorrect, the simulation is aborted.