DWR Technical Memorandum

Integrated Water Flow Model

IWFM-2015

Revision 1129

Programmer's Manual

Emin C. Dogrul and Tariq N. Kadir

Bay-Delta Office, California Department of Water Resources, 1416 Ninth Street, Sacramento, CA 95814

February 2021

DWR Technical Memorandum

Integrated Water Flow Model

IWFM-2015

Revision 1129

Programmer's Manual

Emin C. Dogrul and Tariq N. Kadir

Bay-Delta Office, California Department of Water Resources, 1416 Ninth Street, Sacramento, CA 95814

February 2021

DWR Technical Memorandum: Programmer's Manual for the Integrated Water Flow Model (IWFM-2015), Revision 1129

Authors: Emin C. Dogrul and Tariq N. Kadir

Modeling software and documentation originated and maintained by the Bay-Delta Office, California Department of Water Resources, 1416 Ninth Street, Sacramento, CA 95814

https://www.water.ca.gov/Library/Modeling-and-Analysis/Modeling-Platforms/Integrated-Water-Flow-Model

This report describes programming interfaces for IWFM-2015 Revision 1129, released in February 2021.

Table of Contents

1. Introduction

This document lists procedures exposed through IWFM-2015 Application Programming Interface (API) that can be used by programmers to develop software that accesses input and output data files for IWFM-2015 applications. Currently, IWFM-2015 API allows access to applications' input and output data files for post-processing purposes only and, hence, it does not allow manipulation of input files for IWFM-2015 model application development. It also allows running IWFM model application from within other programs, as well as running, pausing, and continuing to run models. For simplicity, IWFM-2015 will be referred to as IWFM for the rest of this document.

The procedures exposed through the API are tested using Python, Java, C#, Visual Basic and Visual Basic for Applications (VBA) programming languages.

2. Calling Conventions

IWFM API is written using Fortran 2008 programming language. It is compiled for both 32-bit and 64-bit applications running under Microsoft Windows OS. The following approach and data standards are used in the API:

- *stdcall* calling convention is used to allow the API procedures to be called from code written in Visual Basic for Applications (VBA). For instance, this is the case when the API procedures are called from MS Excel.
- All procedure arguments are expected to be passed by reference.
- All procedure arguments are C data types.
- To avoid possible stack overflows, heap memory is used.
- All real number arguments that appear in procedure interfaces are defined as C `double` type; i.e. as 64-bit (8-byte) arguments. In this document, a real type is denoted by `REAL(C_DOUBLE)`.
- All integer arguments that appear in procedure interfaces are defined as C `int` type; i.e. as 32-bit (4-byte) arguments. In this document, an integer type is denoted by `INTEGER(C_INT)`.

- The API allows passing arrays of real and integer arguments. Note that Fortran uses 1-based arrays and it uses column-major order (i.e. first array index changes the fastest) when ordering multi-dimensional arrays. Care must be taken when the API is called from languages that use 0-based arrays and row-major ordering (i.e. last array index changes the fastest).

- When a scalar string argument is passed to an API procedure, it is received as an array of C `char` data type. Both the name of the argument and the number of characters in the string must be passed. In this document, a character type is denoted by **`CHARACTER(C_CHAR)`**.

- String array arguments are not allowed due to complexities in representing strings and arrays of strings in different programming languages. Instead, an array of strings is converted into a string scalar and passed back to the client along with information to convert the scalar string back to a string array.

  As an example, consider the following procedure interface which returns names of model features (subregions, stream reaches, stream gage locations, etc.):

```
SUBROUTINE IW_Model_GetNames(iLocationType,iDimLocArray,iLocArray,  &
                  iLenNamesList,cNamesList,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLocationType,iDimLocArray,  &
                                iLenNamesList
   INTEGER(C_INT),INTENT(OUT) :: iLocArray(iDimLocArray),iStat
   CHARACTER(C_CHAR),INTENT(OUT) :: cNamesList(iLenNamesList)
END SUBROUTINE IW_Model_GetNames
```

  In the above procedure, `cNamesList` is a **`CHARACTER(C_CHAR)`** array with a dimension of `iLenNamesList`. The client code receives it as a string scalar. It stores the contents of a string array with a dimension of `iDimLocArray`. `iLocArray` is an integer array that stores the character positions within the `cNamesList` argument to break it into an array of strings.

  For instance, assume that there are 3 stream gages in an IWFM model application with names

  "Sacramento" (10 characters long)

  "Yuba" (4 characters long)

  "Merced" (6 characters long)

In this case,

`cNamesList` = "SacramentoYubaMerced" (string scalar as received by the client code)

`iLocArray` = [1, 11, 15]

Using `iLocArray`, the client can break `cNamesList` into a string array with a dimension of 3; "Sacramento" starts at character 1 of `cNamesList`, "Yuba" starts at character 11 of `cNamesList` and "Merced" starts at character 15 of `cNamesList`. Note that the client must provide `iDimLocArray` (dimension of the string array) as well as `iLenNamesList` (number of characters in `cNamesList`) that should be large enough to hold all characters that will be stored in `cNamesList`.

3. Language-Specific Calling Mechanisms

In this section, mechanisms specific to different programming languages to call IWFM API procedures will be explained. For this purpose, the following dummy procedures will be used to demonstrate how arguments with different data types are defined in the client programming language and how the API procedure is called. These procedures are included in the IWFM API to test calling mechanisms with other programming languages. Please refer to section 8.4 for the description of these procedures and how to check if the tested calling mechanism works properly.

i. Scalar integer and real numbers (passed to or retrieved from API procedure):

```
SUBROUTINE fooScalar(iArg,dArg)
    INTEGER(C_INT) :: iArg
    REAL(C_DOUBLE) :: dArg
END SUBROUTINE fooScalar
```

ii. 1-dimensional integer and real arrays (passed to or retrieved from API procedure):

```
SUBROUTINE foo1DArray(iArrayDim,iArray,idArrayDim,dArray)
    INTEGER(C_INT) :: iArrayDim,idArrayDim
    INTEGER(C_INT) :: iArray(iArrayDim)
    REAL(C_DOUBLE) :: dArray(idArrayDim)
END SUBROUTINE foo1DArray
```

iii. 2-dimensional integer and real arrays (passed to or retrieved from API procedure):

```
SUBROUTINE foo2DArray(iDim1,iDim2,iArray,idDim1,idDim2,dArray)
    INTEGER(C_INT) :: iDim1,iDim2,idDim1,idDim2
    INTEGER(C_INT) :: iArray(iDim1,iDim2)
    REAL(C_DOUBLE) :: dArray(idDim1,idDim2)
END SUBROUTINE foo2DArray
```

iv. String scalar passed to API procedure:

```
SUBROUTINE fooStrPassed(iLen,cStrPassed)
    INTEGER(C_INT) :: iLen
    CHARACTER(C_CHAR),INTENT(IN) :: cStrPassed(iLen)
END SUBROUTINE fooStrPassed
```

v. String scalar received from the API procedure:

```
SUBROUTINE fooStrReceived(iLen,cStrRecvd)
    INTEGER(C_INT) :: iLen
    CHARACTER(C_CHAR),INTENT(OUT) :: cStrRecvd(iLen)
END SUBROUTINE fooStrReceived
```

3.1. Python

IWFM API procedures are called from Python using the `ctypes` foreign function library. `windll` object exposed by `ctypes` is used to gain access to the API procedures using the *stdcall* calling convention.

```
import ctypes
IWFM_dll = ctypes.windll.LoadLibrary("D:\\IWFM\\Bin\\IWFM2015_C.dll")
```

i. Scalar integer and real numbers:

```
iArg = ctypes.c_int(5)
dArg = ctypes.c_double(3.2)
IWFM_dll.fooScalar(ctypes.byref(iArg), ctypes.byref(dArg))
```

ii. 1-dimensional integer and real arrays:

```
iArrayDim = ctypes.c_int(10)
idArrayDim = ctypes.c_int(15)
iArray = (ctypes.c_int*iArrayDim.value)()
dArray = (ctypes.c_double*idArrayDim.value)()
IWFM_dll.foo1DArray(ctypes.byref(iArrayDim), iArray,  \
    ctypes.byref(idArrayDim), dArray)
```

iii. 2-dimensional integer and real arrays:

```
iDim1 = ctypes.c_int(5)
iDim2 = ctypes.c_int(10)
idDim1 = iDim1
idDim2 = iDim2
i2DArray = ((ctypes.c_int*iDim1.value)*iDim2.value)()
d2DArray = ((ctypes.c_double*idDim1.value)*idDim2.value)()
IWFM_dll.foo2DArray(ctypes.byref(iDim1), ctypes.byref(iDim2),  \
    i2DArray, ctypes.byref(idDim1), ctypes.byref(idDim2), d2DArray)
```

iv. String scalar passed to API procedure:

```
sString = ctypes.create_string_buffer(b"This is a test!")
iLen = ctypes.c_int(ctypes.sizeof(sString))
IWFM_dll.fooStrPassed(ctypes.byref(iLen), sString)
```

v. String scalar received from the API procedure:

```
iLen = ctypes.c_int(50)
sString = ctypes.create_string_buffer(iLen.value)
IWFM_dll.fooStrReceived(ctypes.byref(iLen), sString)
print(sString.value)
```

3.2. Java

IWFM API procedures are accessed from Java using the Java Native Access (JNA) API. JNA provides two types of library mapping: direct and interface mapping. For efficiency, direct mapping is suggested. Individual procedures from the IWFM API are accessed by mapping their signatures directly to a Java native method:

```
import com.sun.jna.Native;
import com.sun.jna.ptr.IntByReference;

public class IWFM {

    static {
        Native.register("IWFM2015_C.dll");
    }

    public static native void fooScalar(IntByReference iArg,
                                        DoubleByReference dArg);

}
```

i. Scalar integer and real numbers (`IntByReference` and `DoubleByReference` classes from the JNA API are used):

```
public static native void fooScalar(IntByReference iArg,
                                     DoubleByReference dArg);

IntByReference iArg = new IntByReference(5);
DoubleByReference dArg = new DoubleByReference(3.2);
IWFM.fooScalar(iArg, dArg);
```

ii. 1-dimensional integer and real arrays (`IntByReference` class from the JNA API is used):

```
public static native void foo1DArray(IntByReference iArrayDim,
                                      int[] iArray,
                                      IntByReference idArrayDim,
                                      double[] dArray);

int[] iArray = new int[10];
double[] dArray = new double[15];
IntByReference iArrayDim = new IntByReference(iArray.length);
IntByReference idArrayDim = new IntByReference(dArray.length);
IWFM.foo1DArray(iArrayDim, iArray, idArrayDim, dArray);
```

iii. 2-dimensional integer and real arrays (the easiest way to pass a 2-dimensional array from Java to Fortran is to flatten it to a 1-dimensional array, keeping in mind that Fortran stores the arrays in column-major order; similarly, a 2-dimensional array can be received from Fortran as a 1-dimensional array and mapped to a 2-dimensional array):

```
public static native void foo2DArray(IntByReference iDim1,
                                      IntByReference iDim2,
                                      int[] iArray,
                                      IntByRference idDim1,
                                      IntByReference idDim2,
                                      double[] dArray);

int iDim1 = 5;
int iDim2 = 10;
int idDim1 = iDim1;
int idDim2 = iDim2;

IntByReference iRefDim1 = new IntByReference(iDim1);
IntByReference iRefDim2 = new IntByReference(iDim2);
IntByReference idRefDim1 = new IntByReference(idDim1);
IntByReference idRefDim2 = new IntByReference(idDim2);
int[] iArray = new int[iDim1*iDim2];
double[] dArray = new double[idDim1*idDim2];
IWFM.foo2DArray(iRefDim1, iRefDim2, iArray,
                idRefDim1, idRefDim2,dArray);

int iRow, iCol, indx;
```

```
int[][] i2DArray=new int[iDim1][iDim2];
indx = 0;
for (iCol=0; iCol < iDim2; iCol++)
     for (iRow=0; iRow < iDim1; iRow++) {
          i2DArray[iRow][iCol]=iArray[indx];
          indx++;
}
double[][] d2DArray =  new double[idDim1][idDim2];
indx = 0;
for (iCol=0; iCol < idDim2; iCol++)
     for (iRow=0; iRow < idDim1; iRow++) {
          d2DArray[iRow][iCol]=dArray[indx];
          indx++;
}
```

iv. String scalar passed to API procedure (`IntByReference` class from the JNA API is used):

```
public static native void fooStrPassed(IntByReference iLen,
                                        String sString);

String sString = "This is a test!";
IntByReference iLen = new IntByReference(sString.length());
IWFM.fooStrPassed(iLen, sString);
```

v. String scalar received from the API procedure (`IntByReference` class from the JNA API is used, the string is received as an array of `byte` and converted to Java String; make sure the string length parameter, `iLen`, is large enough to hold all the characters received):

```
public static native void fooStrReceived(IntByReference iLen,
                                          byte[] bStrRecvd);

IntByReference iLen = new IntByReference(50);
byte[] bss = new byte[iLen.getValue()];
IWFM.fooStrReceived(iLen, bss);
String ss = Native.toString(bss);
```

3.3. C#

IWFM API procedures are accessed from C# using the `DllImportAttribute` class from the `System.Runtime.InteropServices` namespace.

```
using System.Runtime.InteropServices;
```

An example declaration of an IWFM API procedure to be called from C# code is as follows:

```
const string cIWFM2015_DLL = "D:\\IWFM\\Bin\\IWFM2015_C.dll";
[DllImport(cIWFM2015_DLL,
    CallingConvention = CallingConvention.StdCall,
    EntryPoint = "fooScalar",
    CharSet = CharSet.Ansi,
    SetLastError = true,
    ExactSpelling = true)]
public static extern void fooScalar(ref int iArg, ref double dArg);
```

i. Scalar integer and real numbers:

```
[DllImport(cIWFM2015_DLL,
    CallingConvention = CallingConvention.StdCall,
    EntryPoint = "fooScalar",
    CharSet = CharSet.Ansi,
    SetLastError = true,
    ExactSpelling = true)]
public static extern void fooScalar(ref int iArg, ref double dArg);

int iArg = 5;
double dArg = 3.2;
fooScalar(ref iArg, ref dArg);
```

ii. 1-dimensional integer and real arrays (note that only the reference to the first item of each array is passed to the API procedure):

```
[DllImport(cIWFM2015_DLL,
    CallingConvention = CallingConvention.StdCall,
    EntryPoint = "foo1DArray",
    CharSet = CharSet.Ansi,
    SetLastError = true,
    ExactSpelling = true)]
public static extern void foo1DArray(ref int iArrayDim, ref int iArray,
    ref int idArrayDim, ref double dArray);

int iArrayDim = 10;
int idArrayDim = 15;
int[] iArray = new int[iArrayDim];
double[] dArray = new double[idArrayDim];
foo1DArray(ref iArrayDim, ref iArray[0], ref idArrayDim, ref dArray[0]);
```

iii. 2-dimensional integer and real arrays (note that only the reference to the first item of each array is passed to the API procedure. Additionally, since the 2-dimensional arrays below are defined in row-major order, the dimensions must

be reversed for proper operation with the IWFM API; i.e. a 10×5 array in C# is transposed and represented as a 5×10 array in IWFM API):

```
[DllImport(cIWFM2015_DLL,
    CallingConvention = CallingConvention.StdCall,
    EntryPoint = "foo2DArray",
    CharSet = CharSet.Ansi,
    SetLastError = true,
    ExactSpelling = true)]
public static extern void foo2DArray(ref int iDim2, ref int iDim1,
    ref int i2DArray, ref int idDim2, ref int idDim1,
    ref double d2DArray);

int iDim1 = 5;
int iDim2 = 10;
int idDim1 = iDim1;
int idDim2 = iDim2;
int[,] i2DArray = new int[iDim2, iDim1];
double[,] d2DArray = new double[idDim2, idDim1];
foo2DArray(ref iDim1, ref iDim2, ref i2DArray[0,0], ref idDim1,
    ref idDim2, ref d2DArray[0,0]);
```

iv. String scalar passed to API procedure (note that when a `StringBuilder` argument that is already assigned a value is passed to an API procedure, the `Length` property is used to obtain its length in characters):

```
 [DllImport(cIWFM2015_DLL,
    CallingConvention = CallingConvention.StdCall,
    EntryPoint = "fooStrPassed",
    CharSet = CharSet.Ansi,
    SetLastError = true,
    ExactSpelling = true)]
public static extern void fooStrPassed(ref int iLen,
    StringBuilder sString);

StringBuilder sString = new StringBuilder("This is a test!");
int iLen = sString.Length;
fooStrPassed(ref iLen, sString);
```

v. String scalar received from the API procedure (note that when a string argument is received from the API procedure, a `StringBuilder` variable with a long enough capacity is created and the `Capacity` property is used to define its length in characters):

```
[DllImport(cIWFM2015_DLL,
    CallingConvention = CallingConvention.StdCall,
    EntryPoint = "fooStrReceived",
    CharSet = CharSet.Ansi,
    SetLastError = true,
```

```
    ExactSpelling = true)]
public static extern void fooStrReceived(ref int iLen,
    StringBuilder sString);

StringBuilder sString = new StringBuilder(50)
int iLen = sString.Capacity;
fooStrReceived(ref iLen, sString);
```

3.4. Visual Basic

IWFM API procedures are accessed from Visual Basic using the `DllImportAttribute` class from the `System.Runtime.InteropServices` namespace.

```
Imports System.Runtime.InteropServices
```

An example declaration of an IWFM API procedure to be called from Visual Basic code is as follows:

```
Const cIWFM2015_DLL As String = "D:\IWFM\Bin\IWFM2015_C.dll"
<DllImport(cIWFM2015_DLL,
    CallingConvention:=CallingConvention.StdCall,
    EntryPoint:="fooScalar",
    CharSet:=CharSet.Ansi,
    SetLastError:=True,
    ExactSpelling:=True)>
Sub fooScalar(ByRef iArg As Integer, ByRef dArg As Double)
End Sub
```

i. Scalar integer and real numbers:

```
<DllImport(cIWFM2015_DLL,
    CallingConvention:=CallingConvention.StdCall,
    EntryPoint:="fooScalar",
    CharSet:=CharSet.Ansi,
    SetLastError:=True,
    ExactSpelling:=True)>
Public Sub fooScalar(ByRef iArg As Integer, ByRef dArg As Double)
End Sub

Dim iArg As Integer = 5
Dim dArg As Double = 3.2
fooScalar(iArg, dArg)
```

ii. 1-dimensional integer and real arrays (note that only the reference to the first item of each array is passed to the API procedure):

```
<DllImport(cIWFM2015_DLL,
```

```
    CallingConvention:=CallingConvention.StdCall,
    EntryPoint:="foo1DArray",
    CharSet:=CharSet.Ansi,
    SetLastError:=True,
    ExactSpelling:=True)>
Public Sub foo1DArray(ByRef iArrayDim As Integer,
    ByRef iArray As Integer, ByRef idArrayDim As Integer,
    ByRef dArray As Double)
End Sub

Dim iArrayDim As Integer = 10
Dim idArrayDim As Integer = 15
Dim iArray(iArrayDim - 1) As Integer
Dim dArray(idArrayDim - 1) As Double
foo1DArray(iArrayDim, iArray(0), idArrayDim, dArray(0))
```

iii. 2-dimensional integer and real arrays (note that only the reference to the first item of each array is passed to the API procedure. Additionally, since the 2-dimensional arrays below are defined in row-major order, the dimensions must be reversed for proper operation with the IWFM API; i.e. a 10×5 array in Visual Basic is transposed and represented as a 5×10 array in IWFM API):

```
<DllImport(cIWFM2015_DLL,
    CallingConvention:=CallingConvention.StdCall,
    EntryPoint:="foo2DArray",
    CharSet:=CharSet.Ansi,
    SetLastError:=True,
    ExactSpelling:=True)>
Public Sub foo2DArray(ByRef iDim2 As Integer, ByRef iDim1 As Integer,
    ByRef i2DArray As Integer, ByRef idDim2 As Integer,
    ByRef idDim1 As Integer, ByRef d2DArray As Double)
End Sub

Dim iDim1 As Integer = 5
Dim iDim2 As Integer = 10
Dim idDim1 As Integer = iDim1
Dim idDim2 As Integer = iDim2
Dim i2DArray(iDim2 - 1, iDim1 - 1) As Integer
Dim d2DArray(idDim2 - 1, idDim1 - 1) As Double
foo2DArray(iDim1, iDim2, i2DArray(0, 0), idDim1, idDim2, d2DArray(0, 0))
```

iv. String scalar passed to API procedure (note that when a `StringBuilder` argument that is already assigned a value is passed to an API procedure, the `Length` property is used to obtain its length in characters):

```
<DllImport(cIWFM2015_DLL,
    CallingConvention:=CallingConvention.StdCall,
    EntryPoint:="fooStrPassed",
```

```
    CharSet:=CharSet.Ansi,
    SetLastError:=True,
    ExactSpelling=True)>
Public Sub fooStrPassed(ByRef iLen As Integer, sString As StringBuilder)
End Sub

Dim sString As New StringBuilder("This is a test!")
Dim iLen As Integer = sString.Length
fooStrPassed(iLen, sString)
```

v. String scalar received from the API procedure (note that when a string argument is received from the API procedure, a `StringBuilder` variable with a long enough capacity is created and the `Capacity` property is used to define its length in characters):

```
<DllImport(cIWFM2015_DLL,
    CallingConvention:=CallingConvention.StdCall,
    EntryPoint:="fooStrReceived",
    CharSet:=CharSet.Ansi,
    SetLastError:=True,
    ExactSpelling:=True)>
Public Sub fooStrReceived(ByRef iLen As Integer, sString As StringBuilder)
End Sub

Dim sString As New StringBuilder(50)
Dim iLen As Integer = sString.Capacity
fooStrReceived(iLen, sString)
```

3.5. Visual Basic for Applications (VBA)

IWFM API procedures are accessed from VBA using the `Declare` statement. An example declaration of an IWFM API procedure to be called from VBA code is as follows:

```
Public Declare PtrSafe Sub fooScalar Lib "D:\IWFM\Bin\IWFM2015_C.dll" _
    (ByRef iArg As Long, ByRef dArg As Double)
```

i. Scalar integer and real numbers:

```
Public Declare PtrSafe Sub fooScalar Lib "D:\IWFM\Bin\IWFM2015_C.dll" _
    (ByRef iArg As Long, ByRef dArg As Double)

Dim iArg As Long
Dim dArg As Double
iArg = 5
dArg = 3.2
Call fooScalar(iArg, dArg)
```

ii. 1-dimensional integer and real arrays (note that only the reference to the first item of each array is passed to the API procedure):

```
Public Declare PtrSafe Sub foo1DArray Lib "D:\IWFM\Bin\IWFM2015_C.dll" _
    (ByRef iArrayDim As Long, ByRef iArray As Long, _
     ByRef idArrayDim As Long, ByRef dArray As Double)

Dim iArrayDim As Long
Dim idArrayDim As Long
Dim iArray() As Long
Dim dArray() As Double
iArrayDim = 10
idArrayDim = 15
ReDim iArray(iArrayDim - 1)
ReDim dArray(idArrayDim - 1)
Call foo1DArray(iArrayDim, iArray(0), idArrayDim, dArray(0))
```

iii. 2-dimensional integer and real arrays (note that only the reference to the first item of each array is passed to the API procedure):

```
Public Declare PtrSafe Sub foo2DArray Lib "D:\IWFM\Bin\IWFM2015_C.dll" _
    (ByRef iDim2 As Long, ByRef iDim1 As Long, ByRef i2DArray As Long, _
     ByRef idDim2 As Long, ByRef idDim1 As Long, _
     ByRef d2DArray As Double)

Dim iDim1 As Long
Dim iDim2 As Long
Dim idDim1 As Long
Dim idDim2 As Long
Dim i2DArray() As Long
Dim d2DArray() As Double
iDim1 = 5
iDim2 = 10
idDim1 = iDim1
idDim2 = iDim2
ReDim i2DArray(iDim1 - 1, iDim2 - 1)
ReDim d2DArray(idDim1 - 1, idDim2 - 1)
Call foo2DArray(iDim1, iDim2, i2DArray(0, 0), idDim1, _
                idDim2, d2DArray(0, 0)
```

iv. String scalar passed to API procedure:

```
Public Declare PtrSafe Sub fooStrPassed Lib "D:\IWFM\Bin\IWFM2015_C.dll" _
    (ByRef iLen As Long, ByVal sString As String)

Dim sString As String
Dim iLen As Long
sString = "This is a test!"
iLen = Len(sString)
Call fooStrPassed(iLen, sString)
```

v. String scalar received from the API procedure (note that when a string argument is received from the API procedure, a `String` variable with a long enough capacity is created):

```
Public Declare PtrSafe Sub fooStrReceived Lib "D:\IWFM\Bin\IWFM2015_C.dll" _
    (ByRef iLen As Long, ByVal sString As String)

Dim sString As String
Dim iLen As Long
iLen = 50
sString = String(iLen, " ")
Call fooStrReceived(iLen, sString)
```

4. Procedure Groups

The procedures exposed through the IWFM API are grouped into three categories based on their functions:

Model: These are procedures used to retrieve information specific to an IWFM model application as well as to retrieve simulation results.

Budget: These procedures are used to retrieve data from Budget output files generated by an IWFM model application. Data from these files can also be retrieved through *Model* procedures; however, the difference is that *Model* procedures require a Model object to be instantiated first. A Model object contains information about all aspects of an IWFM model application. For large applications, instantiation of a Model object can take a noticeable amount of time. On the other hand, retrieving data directly from Budget files without a Model object instantiation is fast. The downside is that Budget files only contain information about the particular flow process for which they store simulation results for. For instance, Groundwater Budget output file only stores simulated flow terms for the groundwater budget at each subregion of an IWFM model application.

ZBudget: These procedures are used to retrieve data from Z-Budget output files generated by an IWFM model application. Similar to Budget files, data from Z-Budget files can also be retrieved through *Model* procedures by instantiating a Model object. The cons and pros of both approaches are already listed above for the *Budget* procedures.

Miscellaneous: These procedures are used to retrieve information on data type codes used by IWFM numerical engine and IWFM API version numbers. There are also procedures that allow opening and closing of a message file that the API uses to print out error and warning messages. Other utility procedures are also grouped under this category.

5. Pseudocodes for Example Workflows

Depending on the purpose of the client software, procedures from different procedure groups within the IWFM API can be called in different orders. To give the programmer an idea about how the IWFM API can be used, example pseudocodes that use the API are provided below.

5.1. Running an IWFM Model Application from Client Code

It is sometimes necessary to run an IWFM model application from a client code, pause the run, interact with the application (e.g. retrieve some simulation results for real time graphing, update some input parameters such as land use based on transient conditions, etc), then continue simulation from where it was paused.

A pseudocode to run an IWFM model application for while updating the diversions based on another model results is presented below:

i) Initialize the model using the provided filenames for Preprocessor and Simulation Main Files (call `IW_Model_New` procedure)

ii) Advance simulation time one simulation timestep forward (call `IW_Model_AdvanceTime` procedure)

iii) Calculate diversions outside the IWFM model application

iv) Read all time-series input data but overwrite the diversions with those calculated in step iii (call `IW_Model_ReadTSData_Overwrite` procedure)

v) Simulate the hydrologic processes for the timestep (call `IW_Model_SimulateForOneTimeStep` procedure)

vi) Print simulation results to user-specified output files (call `IW_Model_PrintResults` procedure)

vii) Check if the end of simulation period has been reached (call `IW_Model_IsEndOfSimulation` procedure); go to step x if end of simulation period has been reached, otherwise go to next step

viii) Advance the state of the hydrologic system in time (call `IW_Model_AdvanceState` procedure)

ix) Go to step ii

x) Close all input and output files, and clear memory used by the IWFM model application (call `IW_Model_Kill` procedure)

5.2. IWFM Tools Excel Add-in

IWFM Tools Excel Add-in is a client software that allows the users to quickly import IWFM model application results stored in different Budget and Z-Budget files for different flow processes into Excel for analysis. In this section, pseudocode to import data from Budget and Z-Budget files into Excel will be described.

5.2.1. Import Budget Data

The user selects a Budget output file from which data will be imported into Excel. Based on the selected file, the user is provided with the following information:

- A list of location names (e.g. name of subregions) for which data is available
- Budget data column titles
- Simulation beginning and ending dates
- Possible time intervals to import data in

Based on this information, the user selects one or more locations to import data for, one or more budget data columns to be imported as well as time period and data interval. The user can also provide unit conversion factors to convert data from simulation units to analysis units.

The following pseudocode is used in the IWFM Tools Excel Add-in client software:

i) Open Budget output file using the filename provided by the user (call `IW_Budget_OpenFile` procedure)

ii) Retrieve number of timesteps in IWFM simulation (call `IW_Budget_GetNTimeSteps` procedure)

iii) Retrieve simulation timestep, simulation beginning and ending times as well as all the dates in-between based on the simulation timestep (call `IW_Budget_GetTimeSpecs`)

iv) Retrieve number of locations (e.g. number of subregions, stream reaches, lakes, etc. depending on the Budget type) for which data can be imported for (call `IW_Budget_GetNLocations`)

v) Retrieve location names (call `IW_Budget_GetLocationNames`)

vi) Retrieve number of budget data columns (call `IW_Budget_GetNColumns`)

vii) Retrieve the budget data column names (call `IW_Budget_GetColumnHeaders`)

viii) Allow user to select locations, budget data columns, data import period and data import interval

ix) Calculate number of data points to be retrieved for the data import period (e.g. if data import interval is daily, calculate the number of days in the data import period; if monthly interval, calculate number of months) (call `IW_GetNIntervals`)

x) For each location for which budget data will be imported:

x.1) Generate a new Excel worksheet

x.2) Retrieve Excel worksheet titles (call `IW_Budget_GetNTitleLines`, `IW_Budget_GetTitleLength` and `IW_Budget_GetTitleLines`)

x.3) Retrieve data for the selected location and selected budget columns for selected period at selected interval (call `IW_Budget_GetValues`) and store them in Excel worksheet

xi) Close budget file (call `IW_Budget_CloseFile`)

5.2.2. Import Z-Budget Data

The user selects a Z-Budget output file from which data will be imported into Excel. Along with the Z-Budget flow data for each grid cell, the selected file also stores the following information:

- Z-Budget data column titles
- Simulation beginning and ending dates
- Possible time intervals to import data in

The user, then, defines zones by other listing them in Excel or by using a *Zone Definition File* that is also used by the Z-Budget post-processor. Based on this information, the user selects one or more zones to import data for, one or more Z-Budget data columns to be imported as well as time period and data interval. The user can also provide unit conversion factors to convert data from simulation units to analysis units.

The following pseudocode is used in the IWFM Tools Excel Add-in client software:

i) Open Z-Budget output file using the filename provided by the user (call `IW_ZBudget_OpenFile` procedure)

ii) Retrieve number of timesteps in IWFM simulation (call `IW_ZBudget_GetNTimeSteps` procedure)

iii) Retrieve simulation timestep, simulation beginning and ending times as well as all the dates in-between based on the simulation timestep (call `IW_ZBudget_GetTimeSpecs` procedure)

iv) Retrieve number of data columns and column titles that apply to all zones (call `IW_ZBudget_GetColumnHeaders_General` procedure); i.e. if the flow process, which Z-Budget file stores data for, simulates flow exchanges between neighboring zones, all inflows from and outflows to neighboring zones will be listed under two columns, namely "*Inflow from Adjacent*

Zones" and "*Outflow to Adjacent Zones*" (note that this makes sure that all zones have the same number of data columns even though each zone has possibly different number of neighboring zones)

v) Define zones (call `IW_ZBudget_GenerateZoneList_FromFile` procedure if zones are defined in a *Zone Definition File*, or call `IW_ZBudget_GenerateZoneList` procedure if zones are defined within Excel)

vi) Retrieve number of defined zones (call `IW_ZBudget_GetNZones` procedure)

vii) Retrieve zone names (call `IW_ZBudget_GetZoneNames` procedure)

viii) Allow user to select zones, data columns, data import period and data import interval

ix) For each zone for which data will be imported

ix.1) Generate Excel worksheet

ix.2) Retrieve Excel worksheet titles (call `IW_ZBudget_GetNTitles` and `IW_ZBudget_GetTitleLines` procedures)

ix.3) Retrieve zone specific column titles (call `IW_ZBudget_GetColumnHeaders_ForAZone` procedure); in this step column titles for flow exchanges between neighboring zones (if the flow process simulates these flows) are retrieved separately (e.g. "*Inflow from Zone 2*", "*Outflow to Zone 2*", "*Inflow from Zone 7*", "*Outflow to Zone 7*", etc.) as well as column indices to be used to retrieve flow data for these columns

x) Retrieve flow data for selected zones and selected data columns one data interval at a time starting from the selected data import begin date until selected data import end date is reached

x.1) Retrieve data for selected zones and selected columns for the specified data interval starting at current data retrieval date and time (call `IW_ZBudget_GetValues_ForSomeZones_ForAnInterval` procedure)

x.2) Increment current data retrieval date and time by the data interval (call `IW_IncrementTime` procedure)

x.3) If current data retrieval date and time is less than the data retrieval ending date and time (call `IW_IsTimeGreaterThan` procedure) go to step x.1

xi) Close Z-Budget file (call `IW_ZBudget_CloseFile` procedure)

5.3. IWFM ArcGIS GUI

IWFM ArcGIS GUI is an IWFM API client that extends ESRI's ArcMap user interface to retrieve and graph IWFM model application results for map features (subregions, stream reaches, stream nodes, etc.) selected through ArcGIS's map interface. The user specifies the IWFM model application's Preprocessor and Simulation Main Input Files, the projection used for the nodal coordinates, and a name for the project. The GUI then instantiates a Model object and creates all the shapefiles for the IWFM model application through communication with the Model object with the help of the IWFM API. Once the shapefiles are created and displayed, the user can select different map features through the ArcMap user interface. For each selected feature, types of available data are listed. The user can then select any of these data types, retrieve and graph the data at desired intervals.

The following pseudocode is used in this client software:

i) Retrieve all data type codes from IWFM API (call `IW_GetDataUnitTypeIDs` and `IW_GetLocationTypeIDs` procedures)

ii) Set the message file for the IWFM API (call `IW_SetLogFile`)

iii) Instantiate the Model object using the Preprocessor and Simulation Main Input Filenames specified by the user (call `IW_Model_New`)

iv) Retrieve number of finite element nodes and their coordinates (call `IW_Model_GetNNodes` and `IW_Model_GetNodeXY` procedures), and generate nodes point feature class

v) Retrieve number of finite element cells and the nodes that make up each cell (call `IW_Model_GetNElements` and `IW_Model_GetElementConfigData`), and generate elements polygon feature class

vi) Retrieve subregion number that each cell belongs to (call `IW_Model_GetNSubregions` and `IW_Model_GetElemSubregions`), and generate subregions polygon feature class

vii) Similarly, retrieve information from the Model object and generate feature classes for stream nodes, stream reaches, lakes, tile drains, groundwater head hydrograph locations, stream flow hydrograph locations, and subsidence hydrograph locations

viii) When the user selects a feature type (e.g. subregion, stream reach, cell, etc.) through the ArcMap map interface, identify and display the types of time-series data available for that feature (call `IW_Model_GetDataList_AtLocationType`)

ix) When the user selects one of the data types identified in the previous step, check if there are any sub-data related to the selected results type (e.g. groundwater budget results type has individual flow components as sub-data) and display the sub-data, if any (call `IW_GetSubDataList_AtLocation`)

x) When the user selects a data or sub-data type, retrieve the selected time-series data and graph it (call `IW_Model_GetModelData_AtLocation`)

xi) When the user closes the GUI, close message file for the API and terminate the Model object (call `IW_CloseLogFile` and `IW_Model_Kill` procedures)

6. A Note on Instantiating a Model Object

To run, retrieve information about or access the results of a model, the client software first needs to instantiate a Model object (except when Budget or Z-Budget files are accessed directly; in which case a Model object is not instantiated). The Model object is instantiated within the memory space of the IWFM API and is not

directly accessible to the client software. Instead, the client software interacts with the Model object through the procedures available through the IWFM API.

Instantiating a Model object is an expensive process in terms of CPU time and memory usage. Additionally, a Model object may be instantiated by a client software for different reasons: to run a model from beginning to the end uninterrupted; to run a model for a period, pause it, retrieve some information from it, maybe update some stresses based on this information and resume model run; to query the model to obtain information about its structure (e.g. number of nodes, elements, aquifer layers, stream nodes and reaches, etc.); or to access the simulation results of an already run model.

Acknowledging different reasons for instantiating a Model object and to avoid the expensive process of instantiation, IWFM API either instantiates a "complete" or a "partial" Model object. In almost all cases a complete Model object is instantiated except when the Model object is instantiated for query purposes in which case a partial Model object is instantiated. Depending on the size of the IWFM model application, instantiation of a complete Model object can take several minutes while a partial Model object is instantiated in less than a second.

The constructor procedure, `IW_Model_New` (see section 8.1.1), provides the argument `iIsForInquiry` for this purpose. If this argument is set to 1, a partial Model object is instantiated except for the first time when `IW_Model_New` procedure is called (see below for details), otherwise a complete Model object is instantiated.

When `IW_Model_New` procedure is called with `iIsForInquiry` set to 1 to instantiate a Model object for given Pre-processor and Simulation Main Data Files for an IWFM model application for the first time, IWFM API instantiates a complete Model object. During this instantiation process, the API performs two additional tasks:

i) All model simulation results that are stored in an ASCII text or a HEC-DSS files (e.g. groundwater head, stream flow, subsidence and tile drain hydrographs) are transferred to HDF5 files

ii) Some model related information is saved in a file named *IW_ModelData_ForInquiry.bin* in the same folder that the Simulation Main Data File resides. This file includes the following information:

- Simulation period and length of timestep
- Number of unsaturated zone layers
- Number of tile drain nodes
- Number of small watersheds
- Number of groundwater head hydrographs
- Number of stream flow/stage hydrographs
- Number of subsidence hydrographs
- Number of tile drain hydrographs
- Available simulation results and the filenames where these results are stored for groundwater nodes, elements, subregions, lakes, stream nodes, stream reaches, small watersheds as well as filenames for groundwater, stream flow, subsidence and tile drain hydrographs

The next time `IW_Model_New` procedure is called again with `iIsForInquiry` set to 1, IWFM API reads model data from the Pre-processor binary output file and checks for the presence of the *IW_ModelData_ForInquiry.bin* file. Once found, the IWFM API reads the information stored in this file to instantiate a partial Model object. Therefore, a partial Model object includes only the model data listed in the Pre-processor files as well as the information listed in the *IW_ModelData_ForInquiry.bin* file. Trying to retrieve other model-related information (e.g. number of diversions, coordinates of hydrograph print-out locations, number of agricultural crops, etc) from this object will not be successful and the IWFM API will generate a warning message. If retrieval of information that is more than the partial Model object can provide is desired, then `IW_Model_DeleteInquiryDataFile` (see section 8.1.116) procedure can be called to delete the *IW_ModelData_ForInquiry.bin* file before calling the `IW_Model_New` procedure again to force the IWFM API to instantiate a complete Model object.

In general, the following points should be paid attention to when instantiating a Model object:

i) When calling `IW_Model_New` procedure, setting `iIsForInquiry` argument to 0 will cause the deletion of all the model simulation output files and a

complete Model object will be instantiated; this type of Model object instantiation is generally used to run a model

ii) When calling `IW_Model_New` procedure, if `iIsForInquiry` argument is set to 1, IWFM API will check if *IW_ModelData_ForInquiry.bin* file exists in the same folder as the Simulation Main Data File. If this file is found, a partial Model object will be instantiated; otherwise a complete Model object is instantiated and *IW_ModelData_ForInquiry.bin* file will be generated for future instantiations

iii) If detailed model information is required to be retrieved, a complete Model object must be instantiated by making sure that *IW_ModelData_ForInquiry.bin* file does not exist; `IW_Model_DeleteInquiryDataFile` procedure can be used to delete any existing *IW_ModelData_ForInquiry.bin* file

7. IWFM Model Feature Indices versus Identification Numbers

Each IWFM model feature (e.g. cells, nodes, subregions, stream nodes and reaches, diversions, hydrographs, etc.) is assigned an identification number (ID) by the IWFM modeler. These IDs can be any integer number; they don't need to start from 1 or they don't need to be sequential. As an example, a model with 5 subregions can have subregion IDs as

Subregion (1) = 4
Subregion (2) = 5
Subregion (3) = 12
Subregion (4) = 1
Subregion (5) = 7

In the above example, numbers in parentheses are the indices of subregions and 4, 5, 12, 1 and 7 are the IDs. The indices reflect the order in which IWFM stores subregion information in the memory. The IWFM modeler does not have any knowledge of the indices, only the IDs that he/she has assigned to the model features.

All IWFM API procedures require indices of the model features as input arguments and almost all the time return back indices of the features, instead of the ID numbers. For instance, procedure `IW_Model_GetStrmUpstrmNodes` (see section 8.1.20) return the stream node indices that are immediately upstream of a stream node indicated by its own index.

Client software for IWFM API is expected to make the conversions between model feature indices and IDs when necessary. To allow this, IWFM API provides procedures to obtain model feature IDs (e.g. `IW_Model_GetStrmNodeIDs`, `IW_Model_GetElementIDs`, `IW_Model_GetDiversionIDs`, etc.) in integer array format.

As an example, let's assume a client software is designed to ask the user for a subregion ID number to retrieve and plot the groundwater budget data. For the subregion example given above, the client software can call procedure `IW_Model_GetSubregionIDs` and receive an integer array of {4, 5, 12, 1, 7}. This array can be used to convert subregion IDs to indices and vice versa. As an example, subregion ID 12 corresponds to index 3, and all IWFM API procedures require index 3 as an input (if this is a required input argument), instead of ID 12. In fact, passing 12 to the IWFM API will produce a memory error.

8. Procedure Interfaces

8.1. Model Group

These procedures allow the client software to instantiate, interact with and release memory used by a Model object.

8.1.1. IW_Model_New

This procedure instantiates a Model object given the Preprocessor and Simulation Main Input Filenames. It opens all related files and allocates memory for the Model object.

```
SUBROUTINE IW_Model_New(iLenPPFileName,cPPFileName,iLenSimFileName,  &
              cSimFileName,iIsRoutedStreams,iIsForInquiry,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLenPPFileName,iLenSimFileName,  &
```

```
                                iIsRoutedStreams,iIsForInquiry
   CHARACTER(C_CHAR),INTENT(IN) :: cPPFileName(iLenPPFileName),  &
                                   cSimFileName(iLenSimFileName)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_New
```

`iLenPPFileName` : Character length of Preprocessor Main Input Filename

`cPPFileName` : Preprocessor Main Input Filename

`iLenSimFileName` : Character length of Simulation Main Input Filename

`cSimFileName` : Simulation Main Input Filename

`iIsRoutedStreams` : Flag to specify if the stream flows are simulated or if the stream network is defined but stream flows are simulated by a separate model; i.e. IWFM is linked to a separate stream routing model (1 = stream flows are simulated within IWFM; 0 = stream network is defined but stream flows are simulated outside IWFM)

`iIsForInquiry` : Flag to specify if the Model object is instantiated to perform simulation or to retrieve data from an already simulated model; note that instantiating the model to perform simulation will delete results from previous simulation runs (1 = Model object is instantiated to retrieve data; 0 = Model object is instantiated to perform a simulation)

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.2. IW_Model_Kill

This procedure terminates the Model object, closes all files associated with the model and clears memory.

```
SUBROUTINE IW_Model_Kill(iStat)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_Kill
```

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.3. IW_Model_GetCurrentDateAndTime

This procedure retrieves the date and time for which hydrologic flows are being simulated.

```
SUBROUTINE IW_Model_GetCurrentDateAndTime(iLenDateTime,  &
              cCurrentDateAndTime,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLenDateTime
   CHARACTER(C_CHAR),INTENT(OUT) :: cCurrentDateAndTime(iLenDateTime)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_Kill
```

`iLenDateAndTime` : Character length of the simulation date and time

`cCurrentDateAndTime` : Date and time for which hydrologic flows are being simulated; the format is MM/DD/YYYY_hh:mm

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.4. IW_Model_GetNTimeSteps

This procedure retrieves the number of timesteps for within the model simulation period.

```
SUBROUTINE IW_Model_GetNTimeSteps(NTimeSteps,iStat)
   INTEGER(C_INT),INTENT(OUT) :: NTimeSteps,iStat
END SUBROUTINE IW_Model_GetNTimeSteps
```

`NTimeSteps` : Number of timesteps within the model simulation period

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.5. IW_Model_GetTimeSpecs

This procedure retrieves all the timestamps, incremented by the model simulation time interval, from the beginning of the simulation period to the end. Simulation time interval is also returned.

```
SUBROUTINE IW_Model_GetTimeSpecs(cDataDatesAndTimes,iLenDates,cInterval,  &
              iLenInterval,NData,iLocArray,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLenDates,iLenInterval,NData
```

```
   CHARACTER(C_CHAR),INTENT(OUT) :: cDataDatesAndTimes(iLenDates),  &
                                    cInterval(iLenInterval)
   INTEGER,INTENT(OUT) :: iLocArray(NData),iStat
END SUBROUTINE IW_Model_GetTimeSpecs
```

`cDataDatesAndTimes` : All the timestamps, incremented by the model simulation time interval, starting from the beginning date of the simulation to the ending date; all timestamps are concatenated into a scalar string argument

`iLenDates` : Character length of `cDataDatesAndTimes` argument which must be set to 16 times `NData` (see below) or greater

`cInterval` : Simulation timestep

`iLenInterval` : Character length of `cInterval` argument which must be set to 8 or greater

`NData` : Number of timestamps being retrieved; it can be obtained by calling procedure `IW_Model_GetNTimeSteps` (see section 8.1.4)

`iLocArray` : Character position array so that client software can separate the `cDataDatesAndTimes` scalar string argument into simulation beginning and end dates

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.6. IW_Model_GetOutputIntervals

This procedure lists the possible time intervals that a selected time-series data can be retrieved at based on the simulation timestep used in the IWFM model.

```
SUBROUTINE IW_Model_GetOutputIntervals(cOutputIntervals,        &
             iLenOutputIntervals,iLocArray,iDim_LocArray_In,    &
             iDim_LocArray_Out,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLenOutputIntervals, iDim_LocArray_In
   CHARACTER(C_CHAR),INTENT(OUT) :: cOutputIntervals(iLenOutputIntervals)
   INTEGER(C_INT),INTENT(OUT) :: iDim_LocArray_Out,
                                 iLocArray(iDim_LocArray_In),iStat
END SUBROUTINE IW_Model_GetOutputIntervals
```

`cOutputIntervals` : List of time intervals data can be retrieved at; multiple time intervals are concatenated into a scalar string argument

`iLenOutputIntervals` : Character length of the `cOutputIntervals` argument which must be set to 160 or larger

`iLocArray` : Character position array so that client software can separate the `cOutputIntervals` scalar string argument into individual time intervals

`iDim_LocArray_In` : Maximum number of possible time intervals which must be set to 20 or larger

`iDim_LocArray_Out` : Actual number of time intervals that the selected data can be retrieved at

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.7. IW_Model_GetNNodes

This procedure retrieves the number of finite element grid nodes.

```
SUBROUTINE IW_Model_GetNNodes(NNodes,iStat)
    INTEGER(C_INT),INTENT(OUT) :: NNodes,iStat
END SUBROUTINE IW_Model_GetNNodes
```

`NNodes` : Number of finite element grid nodes

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.8. IW_Model_GetNodeXY

This procedure retrieves the coordinates of the finite element grid nodes.

```
SUBROUTINE IW_Model_GetNodeXY(NNodes,X,Y,iStat)
    INTEGER(C_INT),INTENT(IN) :: NNodes
    REAL(C_DOUBLE),INTENT(OUT) :: X(NNodes),Y(NNodes)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetNodeXY
```

NNodes : Number of finite element grid nodes; this value can be retrieved by calling the IW_Model_GetNNodes (see section 8.1.6)procedure.

X : X-coordinates of the grid nodes

Y : Y-coordinates of the grid nodes

iStat : Error code; returns 0 if the procedure call was successful

8.1.9. IW_Model_GetNodeIDs

This procedure retrieves the finite element grid node identification numbers.

```
SUBROUTINE IW_Model_GetNodeIDs(NNodes,IDs,iStat)
   INTEGER(C_INT),INTENT(IN) :: NNodes
   INTEGER(C_INT),INTENT(OUT) :: IDs(NNodes),iStat
END SUBROUTINE IW_Model_GetNodeIDs
```

NNodes : Number of finite element grid nodes; this value can be retrieved by calling the IW_Model_GetNNodes (see section 8.1.6) procedure.

IDs : Node identification numbers specified by the user

iStat : Error code; returns 0 if the procedure call was successful

8.1.10. IW_Model_GetNElements

This procedure retrieves the number of finite element grid cells in the IWFM model application.

```
SUBROUTINE IW_Model_GetNElements(NElem,iStat)
   INTEGER(C_INT),INTENT(OUT) :: NElem,iStat
END SUBROUTINE IW_Model_GetNElements
```

NElem : Number of finite element grid cells

iStat : Error code; returns 0 if the procedure call was successful

8.1.11. IW_Model_GetElementIDs

This procedure retrieves the user-specified identification numbers for the finite element grid cells in the IWFM model application.

```
SUBROUTINE IW_Model_GetElementIDs(NElem,IDs,iStat)
    INTEGER(C_INT),INTENT(IN) :: NElem
    INTEGER(C_INT),INTENT(OUT) :: IDs(NElem),iStat
END SUBROUTINE IW_Model_GetElementIDs
```

`NElem` : Number of finite element grid cells

`IDs` : Element identification numbers specified by the user

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.12. IW_Model_GetElementConfigData

This procedure retrieves the list of grid nodes surrounding a specified cell (i.e. vertices of the cell) in counter-clockwise direction. In IWFM, a grid cell can either be triangular with three vertices or quadrilateral with four vertices. For triangular cells this procedure returns 0 (zero) as the fourth vertex.

```
SUBROUTINE IW_Model_GetElementConfigData(iElem,iDim,Nodes,iStat)
    INTEGER(C_INT),INTENT(IN) :: iElem,iDim
    INTEGER(C_INT),INTENT(OUT) :: Nodes(iDim),iStat
END SUBROUTINE IW_Model_GetElementConfigData
```

`iElem` : Cell index whose vertices are being retrieved

`iDim` : Number of vertices to be retrieved which must be set to 4

`Nodes` : Vertices of the cell `iElem` listed in counter-clockwise direction; for triangular cells fourth vertex will be returned as 0 (zero)

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.13. IW_Model_GetNSubregions

This procedure retrieves the number of subregions within the IWFM model application.

```
SUBROUTINE IW_Model_GetNSubregions(NSubregions,iStat)
   INTEGER(C_INT),INTENT(OUT) :: NSubregions,iStat
END SUBROUTINE IW_Model_GetNSubregions
```

`NSubregions` : Number of subregions

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.14. IW_Model_GetSubregionIDs

This procedure retrieves the user-specified identification numbers for the subregions in the IWFM model application.

```
SUBROUTINE IW_Model_GetSubregionIDs(NSubregion,IDs,iStat)
   INTEGER(C_INT),INTENT(IN) :: NSubregion
   INTEGER(C_INT),INTENT(OUT) :: IDs(NSubregion),iStat
END SUBROUTINE IW_Model_GetSubregionIDs
```

`NSubregion` : Number of model subregions

`IDs` : Subregion identification numbers specified by the user

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.15. IW_Model_GetSubregionName

This procedure retrieves the name of a subregion.

```
SUBROUTINE IW_Model_GetSubregionName(iRegion,iLen,cName,iStat)
   INTEGER(C_INT),INTENT(IN) :: iRegion,iLen
   CHARACTER(C_CHAR),INTENT(OUT) :: cName(iLen)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetSubregionName
```

iRegion	:	Subregion index; total number of subregions in an IWFM model application can be obtained by calling procedure IW_Model_GetNSubregions (see section 8.1.13)
iLen	:	Character length of the subregion name which should be large enough to store the entire name; a value of 50 is appropriate
cName	:	Name of the subregion
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.16. IW_Model_GetElemSubregions

This procedure retrieves the index of the subregion that each grid cell belongs to.

```
SUBROUTINE IW_Model_GetElemSubregions(NElem,ElemSubregions,iStat)
    INTEGER(C_INT),INTENT(IN) :: NElem
    INTEGER(C_INT),INTENT(OUT) :: ElemSubregions(NElem),iStat
END SUBROUTINE IW_Model_GetElemSubregions
```

NElem	:	Total number of grid cells which can be obtained by calling procedure IW_Model_NElements (see section 8.1.10)
ElemSubregions	:	Subregion index that each grid cell is associated with
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.17. IW_Model_GetNStrmNodes

This procedure retrieves the number of stream nodes.

```
SUBROUTINE IW_Model_GetNStrmNodes(NStrmNodes,iStat)
    INTEGER(C_INT),INTENT(OUT) :: NStrmNodes,iStat
END SUBROUTINE IW_Model_GetNStrmNodes
```

NStrmNodes	:	Number of stream nodes
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.18. IW_Model_GetStrmNodeIDs

This procedure retrieves the user-specified identification numbers for the stream nodes in the IWFM model application.

```
SUBROUTINE IW_Model_GetStrmNodeIDs(NStrmNodes,IDs,iStat)
   INTEGER(C_INT),INTENT(IN) :: NStrmNodes
   INTEGER(C_INT),INTENT(OUT) :: IDs(NStrmNodes),iStat
END FUNCTION IW_Model_GetStrmNodeIDs
```

`NStrmNodes` : Number of stream nodes

`IDs` : Stream node identification numbers specified by the user

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.19. IW_Model_GetStrmNUpstrmNodes

This procedure retrieves the number of stream nodes that are immediately upstream of a specified stream node.

```
SUBROUTINE IW_Model_GetStrmNUpstrmNodes(iStrmNode,iNNodes, &
             iStat)
   INTEGER(C_INT),INTENT(IN) :: iStrmNode
   INTEGER(C_INT),INTENT(OUT) :: iNNodes,iStat
END FUNCTION IW_Model_GetStrmNUpstrmNodes
```

`iStrmNode` : Stream node index for which the number of stream nodes that are immediately upstream of it are being retrieved

`iNNodes` : Number of stream nodes that are immediately upstream of the stream node `iStrmNode`

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.20. IW_Model_GetStrmUpstrmNodes

This procedure retrieves the stream node indices that are immediately upstream of a specified stream node.

```
SUBROUTINE IW_Model_GetStrmUpstrmNodes(iStrmNode,iNNodes, &
```

```
            iUpstrmNodes,iStat)
    INTEGER(C_INT),INTENT(IN) :: iStrmNode,iNNodes
    INTEGER(C_INT),INTENT(OUT) :: iUpstrmNodes(iNNodes),iStat
END FUNCTION IW_Model_GetStrmUpstrmNodeIDs
```

iStrmNode	:	Stream node index for which immediately upstream nodes are being retrieved
iNNodes	:	Number of stream nodes that are immediately upstream of the stream node iStrmNode; this value can be obtained by calling procedure IW_Model_GetStrmNUpstrmNodes (see section 8.1.19)
iUpstrmNodes	:	Stream node indices that are immediately upstream of stream node iStrmNode
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.21. IW_Model_GetStrmBottomElevs

This procedure retrieves the stream channel bottom elevations at each stream node.

```
SUBROUTINE IW_Model_GetStrmBottomElevs(NNodes,rElevs,iStat)
    INTEGER(C_INT),INTENT(IN) :: NNodes
    REAL(C_DOUBLE),INTENT(OUT) :: rElevs(NNodes)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmBottomElevs
```

NNodes	:	Number stream nodes; this value can be obtained by calling procedure IW_Model_NStrmNodes (see section 8.1.17)
rElevs	:	Stream channel bottom elevations at each stream node
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.22. IW_Model_GetNRatingTablePoints

This procedure retrieves the number of data points in the stream flow rating table for a stream node.

```
SUBROUTINE IW_Model_GetNRatingTablePoints(iStrmNode,N,iStat)
```

```
    INTEGER(C_INT),INTENT(IN) :: iStrmNode
    INTEGER(C_INT),INTENT(OUT) :: N,iStat
END SUBROUTINE IW_Model_GetNRatingTablePoints
```

`iStrmNode` : Stream node index for which number of rating table data points are retrieved

`N` : Number of rating table data points

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.23. IW_Model_GetStrmRatingTable

This procedure retrieves the stream rating table (stage versus flow) at a specified stream node.

```
SUBROUTINE IW_Model_GetStrmRatingTable(iStrmNode,NPoints,Stage,Flow,iStat)
    INTEGER(C_INT),INTENT(IN) :: iStrmNode,NPoints
    REAL(C_DOUBLE),INTENT(OUT) :: Stage(NPoints),Flow(NPoints)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmRatingTable
```

`iStrmNode` : Stream node index for which stage-versus-flow rating table is retrieved

`NPoints` : Number of rating table data points; this value can be obtained by calling procedure `IW_Model_GetNRatingTablePoints` (see section 8.1.22)

`Stage` : Stream stage values listed in the rating table for stream node `iStrmNode`

`Flow` : Stream flow values listed in the rating table for stream node `iStrmNode` corresponding to the `Stage` values

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.24. IW_Model_GetStrmNInflows

This procedure retrieves the number of stream boundary inflows specified by the user as timeseries input data.

```
SUBROUTINE IW_Model_GetStrmNInflows(iNInflows,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iNInflows,iStat
END SUBROUTINE IW_Model_GetStrmNInflows
```

`iNInflows` : Number of stream boundary inflows

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.25. IW_Model_GetStrmInflowNodes

This procedure retrieves indices of the stream nodes that receive boundary inflows specified by the user as timeseries input data.

```
SUBROUTINE IW_Model_GetStrmInflowNodes(iNInflows,iNodes,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNInflows
    INTEGER(C_INT),INTENT(OUT) :: iNodes(iNInflows),iStat
END SUBROUTINE IW_Model_GetStrmInflowNodes
```

`iNInflows` : Number of stream boundary inflow; this value can be obtained by calling procedure `IW_Model_GetStrmNInflows` (see section 8.1.24)

`iNodes` : Indices of stream nodes that receive boundary inflows

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.26. IW_Model_GetStrmInflowIDs

This procedure retrieves the identification numbers for the stream boundary inflows specified by the user as timeseries input data.

```
SUBROUTINE IW_Model_GetStrmInflowIDs(iNInflows,IDs,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNInflows
    INTEGER(C_INT),INTENT(OUT) :: IDs(iNInflows),iStat
END SUBROUTINE IW_Model_GetStrmInflowIDs
```

`iNInflows` : Number of stream boundary inflow; this value can be obtained by calling procedure `IW_Model_GetStrmNInflowInflows` (see section 8.1.24)

`IDs` : Identification numbers for the stream boundary inflows

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.27. IW_Model_GetStrmInflows_AtSomeInflows

This procedure retrieves user-specified stream boundary inflows at a specified set of inflows listed by their indices.

```
SUBROUTINE IW_Model_GetStrmInflows_AtSomeInflows(iNInflows,iInflows, &
           rConvFactor,rInflows,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNInflows,iInflows(iNInflows)
   INTEGER(C_DOUBLE),INTENT(IN) :: rConvFactor
   INTEGER(C_DOUBLE),INTENT(OUT) :: rInflows(iNInflows)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmInflows_AtSomeInflows
```

`iNInflows` : Number of user-specified stream boundary inflows that are being retrieved

`iInflows` : List of inflow indices for which stream boundary inflows are being retrieved

`rConvFactor` : Conversion factor to convert stream boundary inflows from simulation unit of volume to a desired unit of volume

`rInflows` : Stream boundary inflows at inflow indices `iInflows`

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.28. IW_Model_GetStrmFlow

This procedure retrieves stream flow at a stream node index.

```
SUBROUTINE IW_Model_GetStrmFlow(iStrmNode,rFact,rFlow,iStat)
   INTEGER(C_INT),INTENT(IN) :: iStrmNode
   REAL(C_DOUBLE),INTENT(IN) :: rFact
   REAL(C_DOUBLE),INTENT(OUT) :: rFlow
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmFlow
```

`iStrmNode` : Stream node index for which flow is being retrieved

`rFact` : Conversion factor to convert stream flow from simulation unit of volume to a desired unit of volume

`rFlow` : Stream flow at each stream node `iStrmNode`

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.29. IW_Model_GetStrmFlows

This procedure retrieves stream flows at every stream node.

```
SUBROUTINE IW_Model_GetStrmFlows(iNStrmNodes,rFact,Flows,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNStrmNodes
    REAL(C_DOUBLE),INTENT(IN) :: rFact
    REAL(C_DOUBLE),INTENT(OUT) :: Flows(iNStrmNodes)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmFlows
```

`iNStrmNodes` : Number of stream nodes

`rFact` : Conversion factor to convert stream flows from simulation unit of volume to a desired unit of volume

`Flows` : Stream flows at each stream node

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.30. IW_Model_GetStrmStages

This procedure retrieves stream stages at every stream node.

```
SUBROUTINE IW_Model_GetStrmStages(iNStrmNodes,rFact,Stages,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNStrmNodes
    REAL(C_DOUBLE),INTENT(IN) :: rFact
    REAL(C_DOUBLE),INTENT(OUT) :: Stages(iNStrmNodes)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmStages
```

`iNStrmNodes` : Number of stream nodes

`rFact` : Conversion factor to convert stream stages from simulation unit of length to a desired unit of length

Stages	:	Stream stages at each stream node
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.31. IW_Model_GetStrmTributaryInflows

This procedure retrieves small watershed inflows into each stream node.

```
SUBROUTINE IW_Model_GetStrmTributaryInflows(iNNodes, rConvFactor,  &
             rFlows,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNNodes
   REAL(C_DOUBLE),INTENT(IN) :: rConvFactor
   REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNNodes)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmTributaryInflows
```

iNNodes	:	Total number of stream nodes; this value can be obtained by calling the IW_Model_GetNStrmNodes procedure (see section 8.1.17)
rConvFactor	:	Conversion factor to convert tributary inflows from simulation unit of volume to a desired unit of volume
rFlows	:	Tributary inflows into each stream node
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.32. IW_Model_GetStrmRainfallRunoff

This procedure retrieves rainfall runoff into each stream node.

```
SUBROUTINE IW_Model_GetStrmRainfallRunoff(iNNodes,rConvFactor,  &
             rFlows,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNNodes
   REAL(C_DOUBLE),INTENT(IN) :: rConvFactor
   REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNNodes)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmRainfallRunoff
```

`iNNodes` : Total number of stream nodes; this value can be obtained by calling the `IW_Model_GetNStrmNodes` procedure (see section 8.1.17)

`rConvFactor` : Conversion factor to convert inflows due to rainfall runoff from simulation unit of volume to a desired unit of volume

`rFlows` : Inflows due to rainfall runoff into each stream node

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.33. IW_Model_GetStrmReturnFlows

This procedure retrieves agricultural and urban return flows into each stream node.

```
SUBROUTINE IW_Model_GetStrmReturnFlows(iNNodes,rConvFactor,  &
             rConvFactor,rFlows,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNNodes
   REAL(C_DOUBLE),INTENT(IN) :: rConvFactor
   REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNNodes)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmReturnFlows
```

`iNNodes` : Total number of stream nodes; this value can be obtained by calling the `IW_Model_GetNStrmNodes` procedure (see section 8.1.17)

`rConvFactor` : Conversion factor to convert inflows due to return flows from simulation unit of volume to a desired unit of volume

`rFlows` : Inflows due to return flows into each stream node

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.34. IW_Model_GetStrmTileDrains

This procedure retrieves tile drain flows into each stream node.

```
SUBROUTINE IW_Model_GetStrmTileDrains(iNNodes,rConvFactor,  &
             rFlows,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNNodes
```

```
    REAL(C_DOUBLE),INTENT(IN) :: rConvFactor
    REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNNodes)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmTileDrains
```

`iNNodes` : Total number of stream nodes; this value can be obtained by calling the `IW_Model_GetNStrmNodes` procedure (see section 8.1.17)

`rConvFactor` : Conversion factor to convert tile drain inflows from simulation unit of volume to a desired unit of volume

`rFlows` : Tile drain flows into each stream node

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.35. IW_Model_GetStrmRiparianETs

This procedure retrieves riparian evapotranspiration at each stream node.

```
SUBROUTINE IW_Model_GetStrmRiparianETs(iNNodes,rConvFactor,  &
             rFlows,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNNodes
    REAL(C_DOUBLE),INTENT(IN) :: rConvFactor
    REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNNodes)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmRiparianETs
```

`iNNodes` : Total number of stream nodes; this value can be obtained by calling the `IW_Model_GetNStrmNodes` procedure (see section 8.1.17)

`rConvFactor` : Conversion factor to convert riparian evapotranspiration from simulation unit of volume to a desired unit of volume

`rFlows` : Riparian evapotranspiration at each stream node

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.36. IW_Model_GetStrmGainFromGW

This procedure retrieves gain from groundwater at each stream node.

```
SUBROUTINE IW_Model_GetStrmGainFromGW(iNNodes,rConvFactor,  &
             rFlows,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNNodes
    REAL(C_DOUBLE),INTENT(IN) :: rConvFactor
    REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNNodes)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmGainFromGW
```

`iNNodes` : Total number of stream nodes; this value can be obtained by calling the `IW_Model_GetNStrmNodes` procedure (see section 8.1.17)

`rConvFactor` : Conversion factor to convert gain from groundwater from simulation unit of volume to a desired unit of volume

`rFlows` : Gain from groundwater at each stream node

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.37. IW_Model_GetStrmGainFromLakes

This procedure retrieves gain from lakes at each stream node.

```
SUBROUTINE IW_Model_GetStrmGainFromLakes(iNNodes,rConvFactor,  &
             rFlows,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNNodes
    REAL(C_DOUBLE),INTENT(IN) :: rConvFactor
    REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNNodes)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmGainFromLakes
```

`iNNodes` : Total number of stream nodes; this value can be obtained by calling the `IW_Model_GetNStrmNodes` procedure (see section 8.1.17)

`rConvFactor` : Conversion factor to convert gain from lakes from simulation unit of volume to a desired unit of volume

`rFlows` : Gain from lakes at each stream node

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.38. IW_Model_GetStrmNetBypassInflows

This procedure retrieves net bypass inflows at each stream node.

```
SUBROUTINE IW_Model_GetStrmNetBypassInflows(iNNodes,rConvFactor,  &
             rFlows,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNNodes
   REAL(C_DOUBLE),INTENT(IN) :: rConvFactor
   REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNNodes)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmNetBypassInflows
```

`iNNodes` : Total number of stream nodes; this value can be obtained by calling the `IW_Model_GetNStrmNodes` procedure (see section 8.1.17)

`rConvFactor` : Conversion factor to convert net bypass inflows from simulation unit of volume to a desired unit of volume

`rFlows` : Net bypass inflows at each stream node

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.39. IW_Model_GetStrmActualDiversions_AtSomeDiversions

This procedure retrieves actual diversion amounts for a list of diversions. Actual diversions can be less than the required diversions if stream goes dry at the stream nodes where the diversions are taken from.

```
SUBROUTINE IW_Model_GetStrmActualDiversions_AtSomeDiversions(iNDivs,iDivs, &
             rConvFactor,rDivs,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNDivs,iDivs(iNDivs)
   REAL(C_DOUBLE),INTENT(IN) :: rConvFactor
   REAL(C_DOUBLE),INTENT(OUT) :: rDivs(iNDivs)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStrmActualDiversions_AtSomeDiversions
```

`iNDivs` : Number of diversions for which actual diversion amounts are retrieved

`iDivs` : List of diversion indices for which actual diversions are retrieved; note that diversion indices can be different than the diversion identification numbers used in a model

`rConvFactor` : Conversion factor to convert actual diversions from simulation unit of volume to a desired unit of volume

`rDivs` : Actual diversions at diversions listed in the `iDivs` argument

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.40. IW_Model_GetStrmDiversionsExportNodes

This procedure retrieves the stream node indices where a set of diversions are taken out of.

```
SUBROUTINE IW_Model_GetStrmDiversionsExportNodes(iNDivs,iDivList, &
             iStrmNodeList,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNDivs,iDivList(iNDivs)
   INTEGER(C_INT),INTENT(OUT) :: iStrmNodeList(iNDivs),iStat
END SUBROUTINE IW_Model_GetStrmDiversionsExportNodes
```

`iNDivs` : Number of diversions for which stream nodes where they are taken out of are retrieved

`iDivList` : List of diversion indices for which stream nodes are retrieved; note that diversion indices can be different than diversion identification numbers in a model

`iStrmNodeList` : Stream node indices where the listed diversions are taken out of; note that stream node indices can be different than stream node identification numbers in a model

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.41. IW_Model_GetNReaches

This procedure retrieves the number of stream reaches.

```
SUBROUTINE IW_Model_GetNReaches(NReach,iStat)
```

```
    INTEGER(C_INT),INTENT(OUT) :: NReach,iStat
END SUBROUTINE IW_Model_GetNReaches
```

NReach : Number of stream reaches

iStat : Error code; returns 0 if the procedure call was successful

8.1.42. IW_Model_GetReachIDs

This procedure retrieves the user-specified identification numbers for the stream reaches in the IWFM model application.

```
SUBROUTINE IW_Model_GetReachIDs(iNReaches,IDs,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNReaches
    INTEGER(C_INT),INTENT(OUT) :: IDs(iNReaches),iStat
END SUBROUTINE IW_Model_GetReachIDs
```

iNReaches : Number of stream reaches

IDs : Stream reach identification numbers specified by the user

iStat : Error code; returns 0 if the procedure call was successful

8.1.43. IW_Model_GetReachNNodes

This procedure retrieves the number of stream nodes in a specified stream reach.

```
SUBROUTINE IW_Model_GetReachNNodes(iReach,iReachNNodes,iStat)
    INTEGER(C_INT),INTENT(IN) :: iReach
    INTEGER(C_INT),INTENT(OUT) :: iReachNNodes,iStat
END SUBROUTINE IW_Model_GetReachNNodes
```

iReach : Stream reach index; note that reach indices can be different than reach identification numbers specified in a model

iReachNNodes : Number of stream nodes within stream reach iReach

iStat : Error code; returns 0 if the procedure call was successful

8.1.44. IW_Model_GetReachGWNodes

This procedure retrieves the groundwater node indices corresponding to stream nodes in a specified stream reach listed from upstream to downstream.

```
SUBROUTINE IW_Model_GetReachGWNodes(iReach,NNodes,iGWNodes,iStat)
    INTEGER(C_INT),INTENT(IN) :: iReach,NNodes
    INTEGER(C_INT),INTENT(OUT) :: iGWNodes(NNodes),iStat
END SUBROUTINE IW_Model_GetReachGWNodes
```

`iReach` : Stream reach index

`NNodes` : Number of stream nodes within the reach `iReach`; this value can be obtained by calling procedure `IW_Model_GetReachNNodes` (see section 8.1.43)

`iGWNodes` : Groundwater node indices corresponding to stream nodes within the stream reach `iReach`, listed from upstream to downstream.

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.45. IW_Model_GetReachStrmNodes

This procedure retrieves the stream node indices of a specified stream reach listed from upstream to downstream.

```
SUBROUTINE IW_Model_GetReachStrmNodes(iReach,iNNodes,iStrmNodes,iStat)
    INTEGER(C_INT),INTENT(IN) :: iReach,iNNodes
    INTEGER(C_INT),INTENT(OUT) :: iStrmNodes(iNNodes),iStat
END SUBROUTINE IW_Model_GetReachStrmNodes
```

`iReach` : Stream reach index

`iNNodes` : Number of stream nodes within the reach `iReach`; this value can be obtained by calling procedure `IW_Model_GetReachNNodes` (see section 8.1.43)

`iStrmNodes` : Indices of stream nodes of stream reach `iReach`, listed from upstream to downstream.

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.46. IW_Model_GetReaches_ForStrmNodes

This procedure retrieves the stream reach indices that a list of stream nodes belongs to.

```
SUBROUTINE IW_Model_GetReaches_ForStrmNodes(iNNodes,iStrmNodes, &
              iReaches,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNNodes,iStrmNodes(iNNodes)
   INTEGER(C_INT),INTENT(OUT) :: iReaches(iNNodes),iStat
END FUNCTION IW_Model_GetReaches_ForStrmNodes
```

`iNNodes` : Number of stream nodes for which the reaches that they belong to are being retrieved

`iStrmNodes` : Stream node indices for which indices of reaches that they belong are being retrieved

`iReaches` : Stream reach indices that each stream node belongs to

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.47. IW_Model_GetReachUpstrmNodes

This procedure retrieves the indices of the uppermost stream nodes for each stream reach.

```
SUBROUTINE IW_Model_GetReachUpstrmNodes(NReaches,iNodes,iStat)
   INTEGER(C_INT),INTENT(IN) :: NReaches
   INTEGER(C_INT),INTENT(OUT) :: iNodes(NReaches),iStat
END SUBROUTINE IW_Model_GetReachUpstrmNodes
```

`NReaches` : Number of stream reaches simulated; this value can be obtained by calling procedure `IW_Model_GetNReaches` (see section 8.1.41)

`iNodes` : Uppermost stream node index for each stream reach

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.48. IW_Model_GetReachNUpstrmReaches

This procedure retrieves the number of reaches that are immediately upstream of a specified reach.

```
SUBROUTINE IW_Model_GetReachNUpstrmReaches(iReach,iNReaches,iStat)
    INTEGER(C_INT),INTENT(IN) :: iReach
    INTEGER(C_INT),INTENT(OUT) :: iNReaches,iStat
END SUBROUTINE IW_Model_GetReachNUpstrmReaches
```

`iReach` : Reach index for which number of upstream reaches is being retrieved

`iNReaches` : Number of reaches that are immediately upstream of reach `iReach`

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.49. IW_Model_GetReachUpstrmReaches

This procedure retrieves the indices of the reaches that are immediately upstream of a specified reach.

```
SUBROUTINE IW_Model_GetReachUpstrmReaches(iReach,iNReaches, &
              iUpstrmReaches,iStat)
    INTEGER(C_INT),INTENT(IN) :: iReach,iNReaches
    INTEGER(C_INT),INTENT(OUT) :: iUpstrmReaches(iNReaches),iStat
END SUBROUTINE IW_Model_GetReachUpstrmReaches
```

`iReach` : Index of reach for which upstream reaches are being retrieved

`iNReaches` : Number of reaches that are immediately upstream of the reach `iReach`; this value can be obtained by calling procedure `IW_Model_GetReachNUpstrmReaches` (see section 8.1.48)

`iUpstrmReaches` : Indices of the reaches that are immediately upstream of reach `iReach`

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.50. IW_Model_GetReachDownstrmNodes

This procedure retrieves the downstream stream node index for each stream reach.

```
SUBROUTINE IW_Model_GetReachDownstrmNodes(NReaches,iNodes,iStat)
    INTEGER(C_INT),INTENT(IN) :: NReaches
    INTEGER(C_INT),INTENT(OUT) :: iNodes(NReaches),iStat
END SUBROUTINE IW_Model_GetReachDownstrmNodes
```

`NReaches` : Number of stream reaches simulated; this value can be obtained by calling procedure `IW_Model_GetNReaches` (see section 8.1.41)

`iNodes` : Downstream stream node index for each stream reach

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.51. IW_Model_GetReachOutflowDest

This procedure retrieves the destination index that each stream reach flows into. To find out the type of destination (i.e. lake, another stream node or outside the model domain) that the reaches flow into, it is necessary to call `IW_Model_GetReachOutflowDestType` procedure.

```
SUBROUTINE IW_Model_GetReachOutflowDest(NReaches,iDest,iStat)
    INTEGER(C_INT),INTENT(IN) :: NReaches
    INTEGER(C_INT),INTENT(OUT) :: iDest(NReaches),iStat
END SUBROUTINE IW_Model_GetReachOutflowDest
```

`NReaches` : Number of stream reaches simulated; this value can be obtained by calling procedure `IW_Model_GetNReaches` (see section 8.1.41)

`iDest` : Destination index that each stream reach flows into

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.52. IW_Model_GetReachOutflowDestTypes

This procedure retrieves the destination types (i.e. lake, another stream node or outside the model domain) that each stream reach flows into. The flow destination type codes used by the IWFM API can be obtained by calling the `IW_GetFlowDestTypeIDs` procedure from the *Miscellaneous* procedures group.

```
SUBROUTINE IW_Model_GetReachOutflowDestType(NReaches,iDestType,iStat)
    INTEGER(C_INT),INTENT(IN) :: NReaches
    INTEGER(C_INT),INTENT(OUT) :: iDestType(NReaches),iStat
END SUBROUTINE IW_Model_GetReachOutflowDestType
```

`NReaches` : Number of stream reaches simulated; this value can be obtained by calling procedure `IW_Model_GetNReaches` (see section 8.1.41)

`iDestType` : Destination type that each stream reach flows into; the flow destination type codes can be obtained by calling the `IW_GetFlowDestTypeIDs` procedure from the *Miscellaneous* procedures group

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.53. IW_Model_GetNDiversions

This procedure retrieves the number of simulated surface water diversions.

```
SUBROUTINE IW_Model_GetNDiversions(iNDiversions,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iNDiversions,iStat
END SUBROUTINE IW_Model_GetNDiversions
```

`iNDiversions` : Number of simulated surface water diversions

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.54. IW_Model_GetDiversionIDs

This procedure retrieves the surface water diversion identification numbers specified in the model.

```
SUBROUTINE IW_Model_GetDiversionIDs(iNDiversions,iDiversionIDs,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNDiversions)
   INTEGER(C_INT),INTENT(OUT) :: iDiversionIDs(iNDiversions),iStat
END SUBROUTINE IW_Model_GetDiversionIDs
```

`iNDiversions` : Number of simulated surface water diversions; this value can be obtained by calling procedure `IW_Model_GetNDiversions` (see section 8.1.53)

`iDiversionIDs` : Surface water diversion identification numbers specified in the model

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.55. IW_Model_GetNLakes

This procedure retrieves the number of lakes simulated in the IWFM model application.

```
SUBROUTINE IW_Model_GetNLakes(NLakes,iStat)
   INTEGER(C_INT),INTENT(OUT) :: NLakes,iStat
END SUBROUTINE IW_Model_GetNLakes
```

`NLakes` : Number of lakes simulated

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.56. IW_Model_GetLakeIDs

This procedure retrieves the lake identification numbers assigned by the user.

```
SUBROUTINE IW_Model_GetLakeIDs(NLakes,IDs,iStat)
   INTEGER(C_INT),INTENT(IN) :: NLakes
   INTEGER(C_INT),INTENT(OUT) :: IDs(NLakes),iStat
END SUBROUTINE IW_Model_GetLakeIDs
```

NLakes : Number of lakes simulated; this value can be obtained by calling procedure IW_Model_GetNLakes (see section 8.1.55)

IDs : Lake identification numbers assigned by the user

iStat : Error code; returns 0 if the procedure call was successful

8.1.57. IW_Model_GetNElementsInLake

This procedure retrieves the number of finite element grid cells that make up a specified lake.

```
SUBROUTINE IW_Model_GetNElementsInLake(iLake,NElements,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLake
   INTEGER(C_INT),INTENT(OUT) :: NElements,iStat
END SUBROUTINE IW_Model_GetNElementsInLake
```

iLake : Index of the lake for which number of cells are retrieved

NElements : Number of cells that make up lake iLake

iStat : Error code; returns 0 if the procedure call was successful

8.1.58. IW_Model_GetElementsInLake

This procedure retrieves the indices of cells that make up a specified lake.

```
SUBROUTINE IW_Model_GetElementsInLake(iLake,NElems,Elems,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLake,NElems
   INTEGER(C_INT),INTENT(OUT) :: Elems(NElems),iStat
END SUBROUTINE IW_Model_GetElementsInLake
```

iLake : Index of the lake for which the list of cells is retrieved

NElems : Number of cells that make up lake iLake; this value can be obtained by calling procedure IW_Model_GetNElementsInLake (see section 8.1.57)

Elems : Indices of cells that make up lake iLake

iStat : Error code; returns 0 if the procedure call was successful

8.1.59. IW_Model_GetNTileDrainNodes

This procedure retrieves the number of simulated tile drains.

```
SUBROUTINE IW_Model_GetNTileDrainNodes(NTDNodes,iStat)
   INTEGER(C_INT),INTENT(OUT) :: NTDNodes,iStat
END SUBROUTINE IW_Model_GetNTileDrainNodes
```

`NTDNodes` : Number of tile drains simulated

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.60. IW_Model_GetTileDrainIDs

This procedure retrieves the user-specified identification numbers for the tile drains in the IWFM model application.

```
SUBROUTINE IW_Model_GetTileDrainIDs(NTDNodes,IDs,iStat)
   INTEGER(C_INT),INTENT(IN) :: NTDNodes
   INTEGER(C_INT),INTENT(OUT) :: IDs(NTDNodes),iStat
END SUBROUTINE IW_Model_GetTileDrainIDs
```

`NTDNodes` : Number of tile drains; this value can be obtained by calling procedure `IW_Model_GetNTileDrainNodes` (see section 8.1.59)

`IDs` : Tile drain identification numbers specified by the user

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.61. IW_Model_GetTileDrainNodes

This procedure retrieves the finite element grid nodes that are associated with tile drains.

```
SUBROUTINE IW_Model_GetTileDrainNodes(NTDNodes,TDNodes,iStat)
   INTEGER(C_INT),INTENT(IN) :: NTDNodes
   INTEGER(C_INT),INTENT(OUT) :: TDNodes(NTDNodes),iStat
END SUBROUTINE IW_Model_GetTileDrainNodes
```

NTDNodes : Number of tile drains being simulated; this value can be obtained by calling procedure IW_Model_GetNTileDrainNodes (see section 8.1.59)

TDNodes : Finite element grid nodes that are associated with tile drains

iStat : Error code; returns 0 if the procedure call was successful

8.1.62. IW_Model_GetNLayers

This procedure retrieves the number of aquifer layers simulated.

```
SUBROUTINE IW_Model_GetNLayers(NLayers,iStat)
    INTEGER(C_INT),INTENT(OUT) :: NLayers,iStat
END SUBROUTINE IW_Model_GetNLayers
```

NLayers : Number of aquifer layers simulated

iStat : Error code; returns 0 if the procedure call was successful

8.1.63. IW_Model_GetGSElev

This procedure retrieves ground surface elevations at each grid node.

```
SUBROUTINE IW_Model_GetGSElev(NNodes,GSElev,iStat)
    INTEGER(C_INT),INTENT(IN) :: NNodes
    REAL(C_DOUBLE),INTENT(OUT) :: GSElev(NNodes)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetGSElev
```

NNodes : Number of grid nodes which can be obtained by calling procedure IW_Model_GetNNodes (see section 8.1.6)

GSElev : Ground surface elevation at each grid node

iStat : Error code; returns 0 if the procedure call was successful

8.1.64. IW_Model_GetAquiferTopElev

This procedure retrieves elevations of the top of the aquifer layers at each grid node.

```
SUBROUTINE IW_Model_GetAquiferTopElev(NNodes,NLayers,TopElev,iStat)
    INTEGER(C_INT),INTENT(IN) :: NNodes,NLayers
    REAL(C_DOUBLE),INTENT(OUT) :: TopElev(NNodes,NLayers)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetAquiferTopElev
```

`NNodes` : Number of finite element grid nodes which can be obtained by calling procedure `IW_Model_GetNNodes` (see section 8.1.6)

`NLayers` : Number of aquifer layers simulated which can be obtained by calling procedure `IW_Model_NLayers` (see section 8.1.62)

`TopElev` : Elevations of the top of aquifer layers at each node

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.65. IW_Model_GetAquiferBottomElev

This procedure retrieves elevations of the bottom of the aquifer layers at each grid node.

```
SUBROUTINE IW_Model_GetAquiferBottomElev(NNodes,NLayers,BottomElev,iStat)
    INTEGER(C_INT),INTENT(IN) :: NNodes,NLayers
    REAL(C_DOUBLE),INTENT(OUT) :: BottomElev(NNodes,NLayers)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetAquiferBottomElev
```

`NNodes` : Number of finite element grid nodes which can be obtained by calling procedure `IW_Model_GetNNodes` (see section 8.1.6)

`NLayers` : Number of aquifer layers simulated which can be obtained by calling procedure `IW_Model_NLayers` (see section 8.1.62)

`BottomElev` : Elevations of the bottom of aquifer layers at each node

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.66. IW_Model_GetStratigraphy_AtXYCoordinate

This procedure retrieves the stratigraphy information (ground surface elevation, and the elevations of the top and bottom of the aquifer at each layer) at a location described by x-y coordinates.

```
SUBROUTINE IW_Model_GetStratigraphy_AtXYCoordinate(iNLayers,rX,rY,rGSElev, &
              rTopElevs,rBottomElevs,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNLayers
    REAL(C_DOUBLE),INTENT(IN) :: rX,rY
    REAL(C_DOUBLE),INTENT(OUT) :: rGSElev,rTopElevs(iNLayers),  &
                                  rBottomElevs(iNLayers)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetStratigraphy_AtXYCoordinate
```

`iNLayers` : Number of aquifer layers simulated which can be obtained by calling procedure `IW_Model_GetNLayers` (see section 8.1.62)

`rX` : X-coordinate of the location in the consistent unit of length used in the model

`rY` : Y-coordinate of the location in the consistent unit of length used in the model

`rGSElev` : Ground surface elevation at the location described by the x-y coordinate

`rTopElevs` : Elevation of the top of the aquifer for each layer simulated

`rBottomElevs` : Elevation of the bottom of the aquifer for each layer simulated

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.67. IW_Model_GetAquiferHorizontalK

This procedure retrieves the saturated horizontal hydraulic conductivity of the aquifer at each grid node and layer.

```
SUBROUTINE IW_Model_GetAquiferHorizontalK(NNodes,NLayers,Kh,iStat)
    INTEGER(C_INT),INTENT(IN) :: NNodes,NLayers
    REAL(C_DOUBLE),INTENT(OUT) :: Kh(NNodes,NLayers)
    INTEGER(C_INT),INTENT(OUT) :: iStat
```

```
END SUBROUTINE IW_Model_GetAquiferHorizontalK
```

NNodes : Number of finite element grid nodes which can be obtained by calling procedure IW_Model_GetNNodes (see section 8.1.6)

NLayers : Number of aquifer layers simulated which can be obtained by calling procedure IW_Model_NLayers (see section 8.1.62)

Kh : Horizontal saturated hydraulic conductivity at each node and layer

iStat : Error code; returns 0 if the procedure call was successful

8.1.68. IW_Model_GetAquiferVerticalK

This procedure retrieves the saturated vertical hydraulic conductivity of the aquifer at each grid node and layer.

```
SUBROUTINE IW_Model_GetAquiferVerticalK(NNodes,NLayers,Kv,iStat)
    INTEGER(C_INT),INTENT(IN) :: NNodes,NLayers
    REAL(C_DOUBLE),INTENT(OUT) :: Kv(NNodes,NLayers)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetAquiferVerticalK
```

NNodes : Number of finite element grid nodes which can be obtained by calling procedure IW_Model_GetNNodes (see section 8.1.6)

NLayers : Number of aquifer layers simulated which can be obtained by calling procedure IW_Model_NLayers (see section 8.1.62)

Kv : Vertical saturated hydraulic conductivity at each node and layer

iStat : Error code; returns 0 if the procedure call was successful

8.1.69. IW_Model_GetAquitardVerticalK

This procedure retrieves the saturated vertical hydraulic conductivity of the aquitards at each grid node and layer.

```
SUBROUTINE IW_Model_GetAquitardVerticalK(NNodes,NLayers,Kv,iStat)
    INTEGER(C_INT),INTENT(IN) :: NNodes,NLayers
    REAL(C_DOUBLE),INTENT(OUT) :: Kv(NNodes,NLayers)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetAquitardVerticalK
```

NNodes : Number of finite element grid nodes which can be obtained by calling procedure IW_Model_GetNNodes (see section 8.1.6)

NLayers : Number of aquifer layers simulated which can be obtained by calling procedure IW_Model_NLayers (see section 8.1.62)

Kv : Aquitard vertical saturated hydraulic conductivity at each node and layer

iStat : Error code; returns 0 if the procedure call was successful

8.1.70. IW_Model_GetAquiferSy

This procedure retrieves the specific yield of the aquifer at each grid node and layer.

```
SUBROUTINE IW_Model_GetAquiferSy(NNodes,NLayers,Sy,iStat)
    INTEGER(C_INT),INTENT(IN) :: NNodes,NLayers
    REAL(C_DOUBLE),INTENT(OUT) :: Sy(NNodes,NLayers)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetAquiferSy
```

NNodes : Number of finite element grid nodes which can be obtained by calling procedure IW_Model_GetNNodes (see section 8.1.6)

NLayers : Number of aquifer layers simulated which can be obtained by calling procedure IW_Model_NLayers (see section 8.1.62)

Sy : Specific yield at each node and layer

iStat : Error code; returns 0 if the procedure call was successful

8.1.71. IW_Model_GetAquiferSs

This procedure retrieves the storage coefficient (i.e. specific storage multiplied by the aquifer thickness) of the aquifer at each grid node and layer.

```
SUBROUTINE IW_Model_GetAquiferSs(NNodes,NLayers,Ss,iStat)
   INTEGER(C_INT),INTENT(IN) :: NNodes,NLayers
   REAL(C_DOUBLE),INTENT(OUT) :: Ss(NNodes,NLayers)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetAquiferSs
```

NNodes : Number of finite element grid nodes which can be obtained by calling procedure IW_Model_GetNNodes (see section 8.1.6)

NLayers : Number of aquifer layers simulated which can be obtained by calling procedure IW_Model_NLayers (see section 8.1.62)

Ss : Storage coefficient (i.e. specific storage multiplied by the aquifer thickness) at each node and layer

iStat : Error code; returns 0 if the procedure call was successful

8.1.72. IW_Model_GetAquiferParameters

This procedure retrieves all aquifer parameters (horizontal hydraulic conductivity, aquifer and aquitard vertical hydraulic conductivity, specific yield and storage coefficient) at each grid node and layer.

```
SUBROUTINE IW_Model_GetAquiferParameters(NNodes,NLayers,Kh,AquiferKv, &
                                         AquitardKv,Sy,Ss,iStat)
   INTEGER(C_INT),INTENT(IN) :: NNodes,NLayers
   REAL(C_DOUBLE),INTENT(OUT) :: Kh(NNodes,NLayers),                &
                                 AquiferKv(NNodes,NLayers),         &
                                 AquitardKv(NNodes,NLayers),        &
                                 Sy(NNodes,NLayers),Ss(NNodes,NLayers)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetAquiferParameters
```

NNodes : Number of finite element grid nodes which can be obtained by calling procedure IW_Model_GetNNodes (see section 8.1.6)

NLayers : Number of aquifer layers simulated which can be obtained by calling procedure IW_Model_NLayers (see section 8.1.62)

Kh : Aquifer horizontal saturated hydraulic conductivity at each node and layer

`AquiferKv` : Aquifer vertical saturated hydraulic conductivity at each node and layer

`AquitardKv` : Aquitard vertical saturated hydraulic conductivity at each node and layer

`Sy` : Specific yield at each node and layer

`Ss` : Storage coefficient (i.e. specific storage multiplied by the aquifer thickness) at each node and layer

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.73. IW_Model_GetNHydrographs

This procedure retrieves the number of hydrographs being printed by the IWFM model application for a location type. The location type can be groundwater head observation location, stream flow observation location, subsidence observation location or tile drain observation location. For other location types, this procedure will always return a value of 0 (zero).

```
SUBROUTINE IW_Model_GetNHydrographs(iLocationType,NHydrographs,iStat)
    INTEGER(C_INT),INTENT(IN) :: iLocationType
    INTEGER(C_INT),INTENT(OUT) :: NHydrographs,iStat
END SUBROUTINE IW_Model_GetNHydrographs
```

`iLocationType` : Location type code which can be obtained by calling the `IW_GetLocationTypeIDs` procedure from the *Miscellaneous* procedures group

`NHydrographs` : Number of hydrographs printed for the selected location type, `iLocationType`

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.74. IW_Model_GetHydrographIDs

This procedure retrieves the hydrograph identification numbers assigned by the user for a hydrograph location type. The location type can be groundwater head

observation location, stream flow observation location, subsidence observation location or tile drain observation location. For other location types, this procedure will always return a value of 0 (zero).

```
SUBROUTINE IW_Model_GetHydrographIDs(iLocationType,NHydrographs,IDs,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLocationType,NHydrographs
   INTEGER(C_INT),INTENT(OUT) :: IDs(NHydrographs),iStat
END SUBROUTINE IW_Model_GetHydrographIDs
```

`iLocationType` : Location type code which can be obtained by calling the `IW_GetLocationTypeIDs` procedure from the *Miscellaneous* procedures group

`NHydrographs` : Number of hydrographs printed for the selected location type, `iLocationType`; this value can be obtained by calling procedure `IW_Model_GetNHydrographs` (see section 8.1.73)

`IDs` : Hydrograph identification numbers assigned by the user

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.75. IW_Model_GetHydrographCoordinates

This procedure retrieves the coordinates of the hydrograph print-out locations for a location type, mainly to display on a GIS map. Coordinates will only be returned for hydrographs, if there are any, for groundwater head observation location type, stream flow observation location type, subsidence observation location type or tile drain location type.

```
SUBROUTINE IW_Model_GetHydrographCoordinates(iLocationType,NHydrographs,  &
             X,Y,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLocationType,NHydrographs
   REAL(C_DOUBLE),INTENT(OUT) :: X(NHydrographs),Y(NHydrographs)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetHydrographCoordinates
```

`iLocationType` : Location type code which can be obtained by calling the `IW_GetLocationTypeIDs` procedure from the *Miscellaneous* procedures group

NHydrographs	:	Number of hydrographs for the location type, iLocationType; this value can be retrieved by calling procedure IW_Model_GetNHydrographs (see section 8.1.73)
X	:	X-coordinates of the hydrograph locations
Y	:	Y-coordinates of the hydrograph locations
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.76. IW_Model_GetNAgCrops

This procedure retrieves the number of agricultural crops (both ponded and non-ponded crops) being simulated.

```
SUBROUTINE IW_Model_GetNAgCrops(NAgCrops,iStat)
   INTEGER(C_INT),INTENT(OUT) :: NAgCrops,iStat
END SUBROUTINE IW_Model_GetNAgCrops
```

NAgCrops	:	Number of agricultural crops simulated
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.77. IW_Model_GetSupplyPurpose

This procedure retrieves flags for a specified set of water supplies (diversions, well pumping or element pumping) to check if they serve agricultural demand, urban demand or both. Note that with automatic supply adjustment feature of IWFM, supply purpose may change (e.g. an agricultural diversion can be adjusted to meet urban demand, actively converting it from agricultural water supply to urban water supply). To avoid any confusions, this procedure retrieves the flags for the initial assignment of the supplies before they are adjusted.

```
SUBROUTINE IW_Model_GetSupplyPurpose(iSupplyTypeID,iNSupplies,iSupplies, &
             iAgOrUrban,iStat)
   INTEGER(C_INT),INTENT(IN) :: iSupplyTypeID,iNSupplies,
                                iSupplies(iNSupplies)
   INTEGER(C_INT),INTENT(OUT) :: iAgOrUrban(iNSupplies),iStat
END SUBROUTINE IW_Model_GetSupplyPurpose
```

`iSupplyTypeID` : Supply type identification number used by IWFM API; allowed values are the identification numbers for diversions, well pumping and element pumping which can be obtained by calling `IW_GetSupplyTypeID_Diversion` (see section 8.4.33), `IW_GetSupplyTypeID_Well` (see section 8.4.34)and `IW_GetSupplyTypeID_ElemPump` (see section 8.4.35) methods, respectively.

`iNSupplies` : Number of supplies for which flags are being retrieved to check if they serve agricultural water demand, urban water demand or both.

`iSupplies` : Indices of supplies for which flags are being retrieved.

`iAgOrUrban` : Flag for each supply to check if they serve agricultural water demand, urban water demand or both; 10 = supply serves agricultural water demand; 01 = supply serves urban water demand; 11 = supply serves both agricultural and urban water demand.

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.78. IW_Model_GetSupplyRequirement_Ag

This procedure retrieves the agricultural water supply requirement at a specified set of locations (grid cells or subregions).

```
SUBROUTINE IW_Model_GetSupplyRequirement_Ag(iLocationTypeID,iNLocations, &
             iLocationList,rFactor,rSupplyReq,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLocationTypeID,iNLocations,
                                iLocationList(iNLocations)
   REAL(C_DOUBLE),INTENT(IN) :: rFactor
   REAL(C_DOUBLE),INTENT(OUT) :: rSupplyReq(iNLocations)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetSupplyRequirement_Ag
```

`iLocationTypeID` : Location type identification number used by IWFM API; allowed values are the identification numbers for element and subregion which can be obtained by calling `IW_GetLocationTypeID_Element` (see section 8.4.13) and

IW_GetLocationTypeID_Subregion (see section 8.4.14) methods, respectively.

iNLocations : Number of locations (element or subregion) for which agricultural supply requirements are being retrieved.

iLocationList : Indices of locations (elements or subregions) for which agricultural supply requirements are being retrieved.

rFactor : Conversion factor for the agricultural supply requirement values to convert them from model consistent units to the desired output unit.

rSupplyReq : Agricultural supply requirement at each location listed in the iLocationList argument.

iStat : Error code; returns 0 if the procedure call was successful

8.1.79. IW_Model_GetSupplyRequirement_Urb

This procedure retrieves the urban water supply requirement at a specified set of locations (grid cells or subregions).

```
SUBROUTINE IW_Model_GetSupplyRequirement_Urb(iLocationTypeID,iNLocations, &
             iLocationList,rFactor,rSupplyReq,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLocationTypeID,iNLocations,
                                iLocationList(iNLocations)
   REAL(C_DOUBLE),INTENT(IN) :: rFactor
   REAL(C_DOUBLE),INTENT(OUT) :: rSupplyReq(iNLocations)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetSupplyRequirement_Urb
```

iLocationTypeID : Location type identification number used by IWFM API; allowed values are the identification numbers for element and subregion which can be obtained by calling IW_GetLocationTypeID_Element (see section 8.4.13) and IW_GetLocationTypeID_Subregion (see section 8.4.14) methods, respectively.

iNLocations : Number of locations (element or subregion) for which urban supply requirements are being retrieved.

`iLocationList` : Indices of locations (elements or subregions) for which urban supply requirements are being retrieved.

`rFactor` : Conversion factor for the urban supply requirement values to convert them from model consistent units to the desired output unit.

`rSupplyReq` : Urban supply requirement at each location listed in the `iLocationList` argument.

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.80. IW_Model_GetSupplyShortAtOrigin_Ag

This procedure retrieves the agricultural water supply shortage at the origin for a specified set of supplies (diversions, well pumping or element pumping). In other words, it retrieves the supply shortage at the destinations served by the supplies plus any losses during the conveyance of the supply.

```
SUBROUTINE IW_Model_GetSupplyShortAtOrigin_Ag(iSupplyTypeID,iNSupplies, &
             iSupplyList,rFactor,rSupplyShort,iStat)
   INTEGER(C_INT),INTENT(IN) :: iSupplyTypeID,iNSupplies,
                                iSupplyList(iNSupplies)
   REAL(C_DOUBLE),INTENT(IN) :: rFactor
   REAL(C_DOUBLE),INTENT(OUT) :: rSupplyShort(iNSupplies)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetSupplyShortAtOrigin_Ag
```

`iSupplyTypeID` : Supply type identification number used by IWFM API; allowed values are the identification numbers for diversions, well pumping and element pumping which can be obtained by calling `IW_GetSupplyTypeID_Diversion` (see section 8.4.33), `IW_GetSupplyTypeID_Well` (see section 8.4.34) and `IW_GetSupplyTypeID_ElemePump` (see section 8.4.35) methods, respectively.

`iNSupplies` : Number of supplies (diversion, well pumping or element pumping) for which agricultural supply shortages are being retrieved.

`iSupplyList` : Indices of supplies (diversion, well pumping or element pumping) for which agricultural supply shortages are being retrieved.

`rFactor` : Conversion factor for the agricultural supply shortage values to convert them from model consistent units to the desired output unit.

`rSupplyShort` : Agricultural supply shortage at the origin of the supplies listed in the `iSupplyList` argument.

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.81. IW_Model_GetSupplyShortAtOrigin_Urb

This procedure retrieves the urban water supply shortage at the origin for a specified set of supplies (diversions, well pumping or element pumping). In other words, it retrieves the supply shortage at the destinations served by the supplies plus any losses during the conveyance of the supply.

```
SUBROUTINE IW_Model_GetSupplyShortAtOrigin_Urb(iSupplyTypeID,iNSupplies, &
             iSupplyList,rFactor,rSupplyShort,iStat)
   INTEGER(C_INT),INTENT(IN) :: iSupplyTypeID,iNSupplies,
                                iSuppyList(iNSupplies)
   REAL(C_DOUBLE),INTENT(IN) :: rFactor
   REAL(C_DOUBLE),INTENT(OUT) :: rSupplyShort(iNSupplies)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetSupplyShortAtOrigin_Urb
```

`iSupplyTypeID` : Supply type identification number used by IWFM API; allowed values are the identification numbers for diversions, well pumping and element pumping which can be obtained by calling `IW_GetSupplyTypeID_Diversion` (see section 8.4.33), `IW_GetSupplyTypeID_Well` (see section 8.4.34) and `IW_GetSupplyTypeID_ElemePump` (see section 8.4.35) methods, respectively.

`iNSupplies` : Number of supplies (diversion, well pumping or element pumping) for which urban supply shortages are being retrieved.

`iSupplyList` : Indices of supplies (diversion, well pumping or element pumping) for which urban supply shortages are being retrieved.

`rFactor` : Conversion factor for the urban supply shortage values to convert them from model consistent units to the desired output unit.

`rSupplyShort` : Urban supply shortage at the origin of the supplies listed in the `iSupplyList` argument.

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.82. IW_Model_GetNames

This procedure retrieves names of locations for a selected location type; e.g. subregion names. Not all locations have names. For instance, finite element cells do not have names. In this case no names will be returned.

```
SUBROUTINE IW_Model_GetNames(iLocationType,iDimLocArray, iLocArray,  &
             iLenNamesList,cNamesList,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLocationType,iDimLocArray,iLenNamesList
   INTEGER(C_INT),INTENT(OUT) :: iLocArray(iDimLocArray),iStat
   CHARACTER(C_CHAR),INTENT(OUT) :: cNamesList(iLenNamesList)
END SUBROUTINE IW_Model_GetNames
```

`iLocationType` : Location type code which can be obtained by calling the `IW_GetLocationTypeIDs` procedure from the *Miscellaneous* procedures group

`iDimLocArray` : Number of locations for a location type (e.g. number of subregions simulated if location type is subregion or number of finite element cells if location type is finite element); the number of locations can be retrieved using the relevant procedure (e.g. `IW_Model_GetNSubregions`)

`iLocArray` : Character position array so that client software can separate the `cNamesList` scalar string argument into individual location names

`iLenNamesList` : Character length of the `cNamesList` argument which must be set to a large enough value to store all location names; a value of 20 times the number of locations is appropriate

`cNamesList` : List of location names under the selected location type concatenated into a scalar string argument

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.83. IW_Model_GetBudget_N

This procedure retrieves the number of Budget output files available for data retrieval.

```
SUBROUTINE IW_Model_GetBudget_N(iNBudgets,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iNBudgets,iStat
END SUBROUTINE IW_Model_GetBudget_N
```

`iNBudgets` : Number of Budget output files available for data retrieval

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.84. IW_Model_GetBudget_List

This procedure returns a list of the Budget output files that are available for data retrieval, along with the budget type and the location type (i.e. subregion, lake, stream reach, etc.) for which they are generated for.

```
SUBROUTINE IW_Model_GetBudget_List(iNBudgets,iLocArray,iLenBudgetList,  &
             cBudgetList,iBudgetTypeList,iBudgetLocTypeList, iStat)
    INTEGER(C_INT),INTENT(IN) :: iNBudgets,iLenBudgetList
    INTEGER(C_INT),INTENT(OUT) :: iLocArray(iNBudgets),            &
                                  iBudgetTypeList(iNBudgets),      &
                                  iBudgetLocTypeList(iNBudgets),   &
                                  iStat
    CHARACTER(C_CHAR),INTENT(OUT) :: cBudgetList(iLenBudgetList)
END SUBROUTINE IW_Model_GetBudget_List
```

`iNBudgets` : Number of Budget output files available for data retrieval; use procedure `IW_Model_GetBudget_N` (see section 8.1.83) to retrieve this information

`iLocArray` : Character position array so that client software can separate the `cBudgetList` scalar string argument into an array of Budget filenames

`iLenBudgetList` : Character length of the `cBudgetList` argument which must be set to a large enough value to store all filenames; a value of 3000 is appropriate

`cBudgetList` : List of Budgets from which data can be retrieved, concatenated into a scalar string argument

`iBudgetTypeList` : List of Budget type identification numbers from which data can be retrieved; possible Budget type identification numbers can be retrieved by calling `IW_GetBudgetTypeIDs` procedure (see section 8.4.39)

`iBudgetLocTypeList`: List of Budget location types (e.g. subregion, lake, stream node, etc.) identification numbers for which each Budget output was generated for; possible location type identification numbers can be retrieved by calling `IW_GetLocationTypeIDs` procedure (see section 8.4.24)

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.85. IW_Model_GetBudget_NColumns

This procedure retrieves the number of columns (excluding the *Time* column) associated with a Budget output.

```
SUBROUTINE IW_Model_GetBudget_NColumns(iBudType,iNCols,iStat)
   INTEGER(C_INT),INTENT(IN) :: iBudType
   INTEGER(C_INT),INTENT(OUT) :: iNCols,iStat
END SUBROUTINE IW_Model_GetBudget_NColumns
```

`iBudType` : Budget type for the selected Budget file; available budget types for the model can be retrieved by calling procedure `IW_Model_GetBudgetList` (see section 8.1.84)

`iNCols` : Number of columns (excluding the *Time* column) for the selected Budget file

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.86. IW_Model_GetBudget_ColumnTitles

This procedure retrieves the column titles (excluding the *Time* column) associated with a Budget output.

```
SUBROUTINE IW_Model_GetBudget_ColumnTitles(iBudType,iLenUnit,cUnitAR,     &
             cUnitVL,iNCols,iLocArray,iLenTitles,cColTitles,iStat)
   INTEGER(C_INT),INTENT(IN) :: iBudType, iLenUnit,iNCols,iLenTitles
   CHARACTER(C_CHAR),INTENT(IN) :: cUnitAR(iLenUnit),cUnitVL(iLenUnit)
   CHARACTER(C_CHAR),INTENT(OUT) :: cColTitles(iLenTitles)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetBudget_ColumnTitles
```

`iBudType` : Budget type for the selected Budget file; available budget types for the model can be retrieved by calling procedure `IW_Model_GetBudgetList` (see section 8.1.84)

`iLenUnit` : Character length of the area and volume units that will be included in the Budget column titles

`cUnitAR` : Area unit that will be included in the Budget column titles

`cUnitVL` : Volume unit that will be included in the Budget column titles

`iNCols` : Number of columns (excluding the *Time* column) for the selected Budget file; this value should have been obtained by calling `IW_Model_GetBudget_NColumns` procedure (see section 8.1.85)

`iLocArray` : Character position array so that client software can separate the `cColTitles` scalar string argument into an array of column titles

`iLenTitle` : Character length of the `cColTitles` argument which must be set to a large enough value to store all column titles; a value of 3000 is appropriate

`cColTitles` : List of Budget column titles concatenated into a scalar string argument

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.87. IW_Model_GetBudget_AnnualFlows

This procedure retrieves the annual (for each water year starting from October and ending in September) water budget flows from a specified Budget output file for a specified location (e.g. subregion, stream reach, etc.) along with the description of the flow terms. It should be noted that this procedure does not retrieve all the columns stored in the Budget file, only those that are an active inflow or outflow component of a water budget (some columns such as area of agricultural lands in the Land & Water Use Budget represent informational data that are not part of the water budget).

```
SUBROUTINE IW_Model_GetBudget_AnnualFlows(iBudType,iLocIndex,iLUType,     &
              iSWShedBudCompRZ,iLenDate,cBeginDate,cEndDate,rFactVL,      &
              iNFlows_In,iNTimes_In,rFlows,iNFlows_Out,iNTimes_Out,       &
              iLenFlowNames,cFlowNames,iLocArray,iWaterYear,iStat)
   INTEGER(C_INT),INTENT(IN) :: iBudType,iLocIndex,iLUType,               &
                                iSWShedBudCompRZ,iLenDate,iNFlows_In,     &
                                iNTimes_In,iLenFlowNames
   CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),cEndDate(iLenDate)
   REAL(C_DOUBLE),INTENT(IN) :: rFactVL
   REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNFlows_In,iNTimes_In)
   CHARACTER(C_CHAR),INTENT(OUT) :: cFlowNames(iLenFlowNames)
   INTEGER(C_INT),INTENT(OUT) :: iNFlows_Out,iNTimes_Out,           &
                                 iLocArray(iNFlows_In),             &
                                 iWaterYears(iNFlows_In),iStat
END SUBROUTINE IW_Model_GetBudget_AnnualFlows
```

`iBudType` : Budget type for the selected Budget file; available budget types for the model can be retrieved by calling procedure `IW_Model_GetBudgetList` (see section 8.1.84)

`iLocIndex` : Index of location for which the water budget data is being retrieved

`iLUType` : Land use type identification number for which annual water budget flows are being retrieved (note that some Budget files such as Groundwater Budget will return the same data regardless of the value of this argument); the land use identification numbers can be retrieved by calling the `IW_GetLandUseTypeIDs` procedure from the *Miscellaneous* procedures group (see section 8.4.11)

`iSWShedBudCompRZ` : Small watershed budget files include water budget information for the root zone and groundwater components of each watershed; set this parameter to 1 if water budget data for root zone component of a small watershed is required, otherwise water budget information for the groundwater component will be retrieved (set this parameter to any value if the selected Budget file for data retrieval is not for small watersheds)

`iLenDate` : Character length of the beginning and ending dates of the simulation period for which the monthly average water terms will be retrieved; this argument should be set to a minimum of 16

`cBeginDate` : Beginning date, in IWFM timestamp format, of the simulation period for which water budget data is being retrieved; this should be an October date

`cEndDate` : Ending date, in IWFM timestamp format, of the simulation period for which water budget data is being retrieved; this should be a September date

`rFactVL` : Factor to convert simulation unit of flow to desired unit of flow

`iNFlows_In` : Maximum number of water budget flow components that will be retrieved; this can be set to the number of columns in the

budget file obtained by calling procedure `IW_Model_GetBudget_NColumns` (see section 8.1.85)

`iNTimes_In` : Maximum number of years for which data will be retrieved; this can be set to the number of timesteps in the simulation period which can be obtained by calling procedure `IW_Model_GetNTimeSteps` (see section 8.1.4)

`rFlows` : 2-dimensional array that will be returned with the annual water budget flows; the first dimension of the array is used for the water budget column and the second dimension is used for each water year within the data retrieval period defined by `cBeginDate` and `cEndDate`

`iNFlows_Out` : Actual number of flow terms involved in the water budget; this number excludes columns in the Budget file used as informational columns (e.g. area of non-ponded crops in Land & Water Use Budget file)

`iNTimes_Out` : Actual number of water years for which data was retrieved

`iLenFlowNames` : Character length of the description of water budget flow terms concatenated into a string scalar; a value of 3000 is appropriate

`cFlowNames` : Description of water budget flow terms concatenated into a string scalar

`iLocArray` : Character position array so that client software can separate the `cFlowNames` scalar string argument into an array of water budget flow term descriptions

`iWaterYears` : List of water years for which data was retrieved

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.88. IW_Model_GetBudget_MonthlyAverageFlows

This procedure retrieves the average monthly water budget flows from a specified Budget output file for a specified location (e.g. subregion, stream reach, etc.) along with the description of the flow terms. It should be noted that this procedure does not retrieve the average of all the columns stored in the Budget file, only those that are an active inflow or outflow component of a water budget (some columns such as area of agricultural lands in the Land & Water Use Budget represent informational data that are not part of the water budget).

```
SUBROUTINE IW_Model_GetBudget_MonthlyAverageFlows(iBudType,iLocIndex,      &
              iLUType,iSWShedBudCompRZ,iLenDate,cBeginDate,cEndDate,       &
              rFactVL,iNFlows_In,rFlows,iNFlows_Out,iLenFlowNames,         &
              cFlowNames,iLocArray,iStat)
    INTEGER(C_INT),INTENT(IN) :: iBudType,iLocIndex,iLUType,   &
                                 iSWShedBudCompRZ,iLenDate,iNFlows_In,     &
                                 iLenFlowNames
    CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),cEndDate(iLenDate)
    REAL(C_DOUBLE),INTENT(IN) :: rFactVL
    REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNFlows_In,12)
    CHARACTER(C_CHAR),INTENT(OUT) :: cFlowNames(iLenFlowNames)
    INTEGER(C_INT),INTENT(OUT) :: iNFlows_Out,iLocArray(iNFlows_In),  &
                                  iStat
END SUBROUTINE IW_Model_GetBudget_MonthlyAverageFlows
```

`iBudType` : Budget type for the selected Budget file; available budget types for the model can be retrieved by calling procedure `IW_Model_GetBudgetList` (see section 8.1.84)

`iLocIndex` : Index of location for which the water budget data is being retrieved

`iLUType` : Land use type identification number for which average monthly water budget flows are being retrieved (note that some Budget files such as Groundwater Budget will return the same data regardless of the value of this argument); the land use identification numbers can be retrieved by calling the `IW_GetLandUseTypeIDs` procedure from the *Miscellaneous* procedures group (see section 8.4.11)

`iSWShedBudCompRZ` : Small watershed budget files include water budget information for the root zone and groundwater components

of each watershed; set this parameter to 1 if water budget data for root zone component of a small watershed is required, otherwise water budget information for the groundwater component will be retrieved (set this parameter to any value if the selected Budget file for data retrieval is not for small watersheds)

`iLenDate` : Character length of the beginning and ending dates of the simulation period for which the monthly average water terms will be retrieved; this argument should be set to a minimum of 16

`cBeginDate` : Beginning date, in IWFM timestamp format, of the simulation period for which water budget data is being retrieved; this should be an October date

`cEndDate` : Ending date, in IWFM timestamp format, of the simulation period for which water budget data is being retrieved; this should be a September date

`rFactVL` : Factor to convert simulation unit of flow to desired unit of flow

`iNFlows_In` : Maximum number of water budget flow components that will be retrieved; this can be set to the number of columns in the budget file obtained by calling procedure `IW_Model_GetBudget_NColumns` (see section 8.1.85)

`rFlows` : 2-dimensional array that will be returned with the monthly average water budget flows; the first dimension of the array is used for the water budget column and the second dimension is used for each month starting with October and ending with September

`iNFlows_Out` : Actual number of flow terms involved in the water budget; this number excludes columns in the Budget file used as informational columns (e.g. area of non-ponded crops in Land & Water Use Budget file)

`iLenFlowNames` : Character length of the description of water budget flow terms concatenated into a string scalar; a value of 3000 is appropriate

`cFlowNames` : Description of water budget flow terms concatenated into a string scalar

`iLocArray` : Character position array so that client software can separate the `cFlowNames` scalar string argument into an array of water budget flow term descriptions

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.89. IW_Model_GetBudget_TSData

This procedure retrieves time-series simulation data from a selected Budget file at a selected location (subregion, stream node, stream reach, etc.) with a specified time interval for a specified time period. The client can provide unit conversion factors to convert the retrieved time-series data from the simulation units to a different unit. The procedure also retrieves information about the type of the time-series data units; i.e. if the retrieved time-series data is length type, area type or volume type.

```
SUBROUTINE IW_Model_GetBudget_TSData(iBudType,iLocIndex,iNCols,iCols,      &
             iLenDate,cBeginDate,cEndDate,iLenInterval,cInterval,          &
             rFactAR,rFactVL,rOutputDates,iNTimes_In,rOutputValues,        &
             iDataTypes,iNTimes_Out,iStat)
   INTEGER(C_INT),INTENT(IN) :: iBudgetType,iLocationIndex,iNCols,         &
                         iCols(iNCols),iLenDate, iLenInterval,iNTimes_In
   CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),                   &
                         cEndDate(iLenDate),cInterval(iLenInterval)
   REAL(C_DOUBLE),INTENT(IN) :: rFactAR,rFactVL
   REAL(C_DOUBLE),INTENT(OUT) :: rOutputDates(iNTimes_In),                 &
                                 rOutputValues(iNTimes_In,iNCols)
   INTEGER(C_INT),INTENT(OUT) :: iDataTypes(iNCols),iNTimes_Out,iStat
END SUBROUTINE IW_Model_GetBudget_TSData
```

`iBudType` : Budget type for the selected Budget file; available budget types for the model can be retrieved by calling procedure `IW_Model_GetBudgetList` (see section 8.1.84)

`iLocIndex` : Index of location for which the timeseries data is being retrieved

`iNCols` : Number of timeseries columns for which data will be retrieved

`iCols` : List of selected column numbers for which timeseries data will be retrieved; the full list of columns can be retrieved by calling procedure `IW_Model_GetBudget_ColumnTitles` (see section 8.1.86)

`iLenDate` : Character length of the beginning and ending dates of the simulation period for which the monthly average water terms will be retrieved; this argument should be set to a minimum of 16

`cBeginDate` : Beginning date for data retrieval in IWFM timestamp format

`cEndDate` : Ending date for data retrieval in IWFM timestamp format

`iLenInterval` : Character length of the time interval at which timeseries data will be retrieved

`cInterval` : Time interval of the retrieved timeseries data (e.g. retrieving groundwater pumping values with monthly interval when a model was run with a daily timestep); a list of possible time intervals for data retrieval can be obtained by calling procedure `IW_Model_GetOutputIntervals` (see section 8.1.6)

`rFactAR` : Factor to convert simulation unit of area to desired unit of area

`rFactVL` : Factor to convert simulation unit of flow to desired unit of flow

`rOutputDates` : Array of dates corresponding to each retrieved data point given in Julian format; Julian day 1 in this format corresponds to January 1, 1900 to comply with MS Excel Julian format

`iNTimes_In` : Maximum number of timesteps for which data will be retrieved; this can be set to the number of timesteps in the

simulation period which can be obtained by calling procedure IW_Model_GetNTimeSteps (see section 8.1.4)

rOutputValues : 2-dimensional array that will be returned with the retrieved timeseries data; first dimension is the counter for timesteps and the second dimension is for data columns

iNTimes_Out : Actual number of timesteps for which data was retrieved

iDataTypes : Timeseries data unit type (length, area or volume) for each data column that is being returned; data unit type codes used by the IWFM API can be retrieved using IW_GetDataUnitTypeIDs procedure from the *Miscellaneous* procedures group (see section 8.4.4)

iStat : Error code; returns 0 if the procedure call was successful

8.1.90. IW_Model_GetBudget_CumGWStorChange

This procedure retrieves the cumulative change in groundwater storage from the Groundwater Budget file for a selected subregion with a specified time interval for a specified time period. The client can provide unit conversion factors to convert the retrieved data from the simulation units to a different unit.

```
SUBROUTINE IW_Model_GetBudget_CumGWStorChange(iSubrgIndex,iLenDate,       &
             cBeginDate,cEndDate,iLenInterval,cInterval,rFactVL,          &
             rOutputDates,iNTimes_In,rCumGWStorChange,iNTimes_Out,iStat)
   INTEGER(C_INT),INTENT(IN) ::iSubrgIndex,iLenDate,iLenInterval,iNTimes_In
   CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),                  &
                        cEndDate(iLenDate),cInterval(iLenInterval)
   REAL(C_DOUBLE),INTENT(IN) :: rFactVL
   REAL(C_DOUBLE),INTENT(OUT) :: rOutputDates(iNTimes_In),                &
                                 rCumGWStorChange(iNTimes_In)
   INTEGER(C_INT),INTENT(OUT) :: iNTimes_Out,iStat
END SUBROUTINE IW_Model_GetBudget_CumGWStorChange
```

iSubrgIndex : Index of the subregion for which the cumulative change in groundwater storage data is being retrieved

iLenDate : Character length of the beginning and ending dates of the simulation period for which the cumulative change in

groundwater storage will be retrieved; this argument should be set to a minimum of 16

`cBeginDate` : Beginning date for data retrieval in IWFM timestamp format

`cEndDate` : Ending date for data retrieval in IWFM timestamp format

`iLenInterval` : Character length of the time interval at which timeseries data will be retrieved

`cInterval` : Time interval of the retrieved data (e.g. retrieving the data with monthly interval when a model was run with a daily timestep); a list of possible time intervals for data retrieval can be obtained by calling procedure `IW_Model_GetOutputIntervals` (see section 8.1.6)

`rFactVL` : Factor to convert simulation unit of groundwater storage to desired unit of storage

`rOutputDates` : Array of dates corresponding to each retrieved data point given in Julian format; Julian day 1 in this format corresponds to January 1, 1900 to comply with MS Excel Julian format

`iNTimes_In` : Maximum number of timesteps for which data will be retrieved; this can be set to the number of timesteps in the simulation period which can be obtained by calling procedure `IW_Model_GetNTimeSteps` (see section 8.1.4)

`rCumGWStorChange` : Array that will be returned with the retrieved cumulative change in groundwater storage data

`iNTimes_Out` : Actual number of timesteps for which data was retrieved

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.91. IW_Model_GetBudget_AnnualCumGWStorChange

This procedure retrieves annual cumulative change in groundwater storage from the Groundwater Budget file for a selected subregion for a specified time period. Annual

values represent each water year that starts on October 1st of a year and ends on September 30th of the following year. This procedure adjusts, if needed, the beginning and ending of the data retrieval period specified by the client so that the data can be properly compiled for water years. The client can provide unit conversion factors to convert the retrieved data from the simulation units to a different unit.

```
SUBROUTINE IW_Model_GetBudget_AnnualCumGWStorChange(iSubrgIndex,iLenDate,  &
              cBeginDate,cEndDate,rFactVL,iNTimes_In,rCumGWStorChange,     &
              iWaterYears,iNTimes_Out,iStat)
    INTEGER(C_INT),INTENT(IN) ::iSubrgIndex,iLenDate, iNTimes_In
    CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),cEndDate(iLenDate)
    REAL(C_DOUBLE),INTENT(IN) :: rFactVL
    REAL(C_DOUBLE),INTENT(OUT) :: rCumGWStorChange(iNTimes_In)
    INTEGER(C_INT),INTENT(OUT) :: iWaterYears(iNTimes_In),iNTimes_Out,iStat
END SUBROUTINE IW_Model_GetBudget_AnnualCumGWStorChange
```

`iSubrgIndex` : Index of the subregion for which the annual cumulative change in groundwater storage data is being retrieved

`iLenDate` : Character length of the beginning and ending dates of the simulation period for which the annual cumulative change in groundwater storage will be retrieved; this argument should be set to a minimum of 16

`cBeginDate` : Beginning date for data retrieval in IWFM timestamp format

`cEndDate` : Ending date for data retrieval in IWFM timestamp format

`rFactVL` : Factor to convert simulation unit of groundwater storage to desired unit of storage

`iNTimes_In` : Maximum number of timesteps for which data will be retrieved; this can be set to the number of timesteps in the simulation period which can be obtained by calling procedure `IW_Model_GetNTimeSteps` (see section 8.1.4)

`rCumGWStorChange` : Array that will be returned with the retrieved annual cumulative change in groundwater storage data

`iWaterYears` : List of water years for which data was retrieved

`iNTimes_Out` : Actual number of timesteps for which data was retrieved

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.92. IW_Model_GetZBudget_N

This procedure retrieves the number of Z-Budget output files available for data retrieval.

```
SUBROUTINE IW_Model_GetZBudget_N(iNZBudgets,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iNZBudgets,iStat
END SUBROUTINE IW_Model_GetZBudget_N
```

`iNZBudgets` : Number of Z-Budget output files available for data retrieval

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.93. IW_Model_GetZBudget_List

This procedure returns a list of the Z-Budget output files along with their type identification numbers that are available for data retrieval.

```
SUBROUTINE IW_Model_GetZBudget_List(iNZBudgets,iLocArray,iLenZBudgetList,  &
             cZBudgetList,iZBudgetTypeList,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNZBudgets,iLenZBudgetList
   INTEGER(C_INT),INTENT(OUT) :: iLocArray(iNZBudgets),    &
                                 iZBudgetTypeList(iNZBudgets),iStat
   CHARACTER(C_CHAR),INTENT(OUT) :: cZBudgetList(iLenZBudgetList)
END SUBROUTINE IW_Model_GetZBudget_List
```

`iNZBudgets` : Number of Z-Budget output files available for data retrieval; use procedure `IW_Model_GetZBudget_N` (see section 8.1.92) to retrieve this information

`iLocArray` : Character position array so that client software can separate the `cZBudgetList` scalar string argument into an array of Z-Budget filenames

`iLenZBudgetList` : Character length of the `cZBudgetList` argument which must be set to a large enough value to store all filenames; a value of 3000 is appropriate

`cZBudgetList` : List of Z-Budgets from which data can be retrieved, concatenated into a scalar string argument

`iZBudgetTypeList` : List of Z-Budget type identification numbers from which data can be retrieved; possible Z-Budget type identification numbers can be retrieved by calling `IW_GetZBudgetTypeIDs` procedure (see section 8.4.40)

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.94. IW_Model_GetZBudget_NColumns

This procedure retrieves the number of columns (excluding the *Time* column) associated with a Z-Budget output and a given zone identification number.

```
SUBROUTINE IW_Model_GetZBudget_NColumns(iZBudgetType,iZoneID,iZExtent,    &
              iDimZones,iElems,iLayers,iZoneIDs,iNCols,iStat)
    INTEGER(C_INT),INTENT(IN) :: iZBudgetType,iZoneID,iZExtent,            &
                            iDimZones,iElems(iDimZones),iLayers(iDimZones), &
                            iZoneIDs(iDimZones)
    INTEGER(C_INT),INTENT(OUT) :: iNCols,iStat
END SUBROUTINE IW_Model_GetZBudget_NColumns
```

`iZBudgetType` : Z-Budget type for the selected Z-Budget file; available Z-Budget types for the model can be retrieved by calling procedure `IW_Model_GetZBudget_List` (see section 8.1.93)

`iZoneID` : Identification number of the selected zone for which number of columns from the selected Z-Budget file is being retrieved

`iZExtent` : Extent of the zone definition (horizontal or vertical; zone definition codes can be retrieved by calling `IW_GetZoneExtentID_***` procedures from the *Miscellaneous* procedures group)

`iDimZones` : Number of model cells for which a zone number is defined

`iElems` : Array of element numbers for which a zone number is specified

`iLayers` : Array of model layers where elements listed in the `iElems` argument reside; if `iZExtent` is set for horizontal zone definition, components of this array can all be set to 1

`iZoneIDs` : Array of zone numbers defined for the elements listed in the `iElems` array

`iNCols` : Number of columns (excluding the *Time* column) for the selected Z-Budget file and zone

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.95. IW_Model_GetZBudget_ColumnTitles

This procedure retrieves the column titles (excluding the *Time* column) associated with a Z-Budget output and a given zone identification number.

```
SUBROUTINE IW_Model_GetZBudget_ColumnTitles(iZBudgetType,iZoneID,         &
              iZExtent,iDimZones,iElems,iLayers,iZoneIDs,iLenUnit,        &
              cUnitAR,cUnitVL,iNCols,iLocArray,iLenTitles,cColTitles,iStat)
    INTEGER(C_INT),INTENT(IN) :: iZBudgetType,iZoneID,iZExtent,            &
                            iDimZones,iElems(iDimZones),iLayers(iDimZones), &
                            iZoneIDs(iDimZones),iLenUnit,iNCols,iLenTitles
    CHARACTER(C_CHAR),INTENT(IN) :: cUnitAR(iLenUnit),cUnitVL(iLenUnit)
    CHARACTER(C_CHAR),INTENT(OUT) :: cColTitles(iLenTitles)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetZBudget_ColumnTitles
```

`iZBudgetType` : Z-Budget type for the selected Z-Budget file; available Z-Budget types for the model can be retrieved by calling procedure `IW_Model_GetZBudget_List` (see section 8.1.93)

`iZoneID` : Identification number of the selected zone for which column titles from the selected Z-Budget file is being retrieved

`iZExtent` : Extent of the zone definition (horizontal or vertical; zone definition codes can be retrieved by calling

IW_GetZoneExtentID_*** procedures from the *Miscellaneous* procedures group)

iDimZones : Number of model cells for which a zone number is defined

iElems : Array of element numbers for which a zone number is specified

iLayers : Array of model layers where elements listed in the iElems argument reside; if iZExtent is set for horizontal zone definition, components of this array can all be set to 1

iZoneIDs : Array of zone numbers defined for the elements listed in the iElems array

iLenUnit : Character length of the area and volume units that will be included in the Z-Budget column titles

cUnitAR : Area unit that will be included in the Z-Budget column titles

cUnitVL : Volume unit that will be included in the Z-Budget column titles

iNCols : Number of columns (excluding the *Time* column) for the selected Z-Budget file and zone; this value should have been obtained by calling IW_Model_GetZBudget_NColumns procedure (see section 8.1.94)

iLocArray : Character position array so that client software can separate the cColTitles scalar string argument into an array of column titles

iLenTitle : Character length of the cColTitles argument which must be set to a large enough value to store all column titles; a value of 3000 is appropriate

cColTitles : List of Z-Budget column titles for the selected zone, concatenated into a scalar string argument

iStat : Error code; returns 0 if the procedure call was successful

8.1.96. IW_Model_GetZBudget_TSData

This procedure retrieves time-series simulation data from a selected Z-Budget file at a selected zone with a specified time interval for a specified time period. The client can provide unit conversion factors to convert the retrieved time-series data from the simulation units to a different unit. The procedure also retrieves information about the type of the time-series data units; i.e. if the retrieved time-series data is length type, area type or volume type.

```
SUBROUTINE IW_Model_GetZBudget_TSData(iZBudgetType,iZoneID,iNCols,          &
             iCols,iZExtent,iNZones,iElems,iLayers,iZoneIDs,iLenDate,       &
             cBeginDate,cEndDate,iLenInterval,cInterval,rFactAR,            &
             rFactVL,rOutputDates,iNTimes_In,rOutputValues,iDataTypes,      &
             iNTimes_Out,iStat)
   INTEGER(C_INT),INTENT(IN) :: iZBudgetType,iZoneID,iNCols,                &
                         iCols(iNCols),iZExtent,iNZones,iElems(iNZones),    &
                         iLayers(iNZones),iZoneIDs(iNZones),iLenDate,       &
                         iLenInterval,iNTimes_In
   CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),                    &
                         cEndDate(iLenDate),cInterval(iLenInterval)
   REAL(C_DOUBLE),INTENT(IN) :: rFactAR,rFactVL
   REAL(C_DOUBLE),INTENT(OUT) :: rOutputDates(iNTimes_In),                  &
                                 rOutputValues(iNTimes_In,iNCols)
   INTEGER(C_INT),INTENT(OUT) :: iDataTypes(iNCols),iNTimes_Out,iStat
END SUBROUTINE IW_Model_GetZBudget_TSData
```

`iZBudgetType` : Z-Budget type for the selected Z-Budget file; available Z-Budget types for the model can be retrieved by calling procedure `IW_Model_GetZBudget_List` (see section 8.1.93)

`iZoneID` : Identification number of the selected zone for which monthly average water budget flows from the selected Z-Budget file is being retrieved

`iNCols` : Number of timeseries columns for which data will be retrieved

`iCols` : List of column numbers for which timeseries data will be retrieved; the list of columns can be retrieved by calling procedure `IW_Model_GetZBudget_ColumnTitles` (see section 8.1.95)

`iZExtent` : Extent of the zone definition (horizontal or vertical; zone definition codes can be retrieved by calling

`IW_GetZoneExtentID_***` procedures from the *Miscellaneous* procedures group)

`iNZones` : Number of model cells for which a zone number is defined

`iElems` : Array of element numbers for which a zone number is specified

`iLayers` : Array of model layers where elements listed in the `iElems` argument reside; if `iZExtent` is set for horizontal zone definition, components of this array can all be set to 1

`iZoneIDs` : Array of zone numbers defined for the elements listed in the `iElems` array

`iLenDate` : Character length of the beginning and ending dates of the simulation period for which the monthly average water terms will be retrieved; this argument should be set to a minimum of 16

`cBeginDate` : Beginning date for data retrieval in IWFM timestamp format

`cEndDate` : Ending date for data retrieval in IWFM timestamp format

`iLenInterval` : Character length of the time interval at which timeseries data will be retrieved

`cInterval` : Time interval of the retrieved timeseries data (e.g. retrieving groundwater pumping values with monthly interval when a model was run with a daily timestep); a list of possible time intervals for data retrieval can be obtained by calling procedure `IW_Model_GetOutputIntervals` (see section 8.1.6)

`rFactAR` : Factor to convert simulation unit of area to desired unit of area

`rFactVL` : Factor to convert simulation unit of flow to desired unit of flow

`rOutputDates` : Array of dates corresponding to each retrieved data point given in Julian format; Julian day 1 in this format corresponds to January 1, 1900 to comply with MS Excel Julian format

`iNTimes_In` : Maximum number of timesteps for which data will be retrieved; this can be set to the number of timesteps in the simulation period which can be obtained by calling procedure `IW_Model_GetNTimeSteps` (see section 8.1.4)

`rOutputValues` : 2-dimensional array that will be returned with the retrieved timeseries data; first dimension is the counter for timesteps and the second dimension is for data columns

`iNTimes_Out` : Actual number of timesteps for which data was retrieved

`iDataTypes` : Timeseries data unit type (length, area or volume) for each data column that is being returned; data unit type codes used by the IWFM API can be retrieved using `IW_GetDataUnitTypeIDs` procedure from the *Miscellaneous* procedures group (see section 8.4.4)

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.97. IW_Model_GetZBudget_AnnualFlows

This procedure retrieves the annual (for each water year starting from October and ending in September) water budget flows from a specified Z-Budget output file for a specified zone along with the description of the flow terms. It should be noted that this procedure does not retrieve all the columns stored in the Z-Budget file, only those that are an active inflow or outflow component of a water budget (some columns such as area of rice fields in the Land & Water Use Z-Budget represent informational data that are not part of the water budget).

```
SUBROUTINE IW_Model_GetZBudget_AnnualFlows(iZBudgetType,iZoneID,iLUType,   &
             iZExtent,iNZones,iElems,iLayers,iZoneIDs,iLenDate,cBeginDate,&
             cEndDate,rFactVL,iNFlows_In,iNTimes_In,rFlows,iNFlows_Out,   &
             iNTimes_Out,iLenFlowNames,cFlowNames,iLocArray,iWaterYears,  &
             iStat)
   INTEGER(C_INT),INTENT(IN) :: iZBudgetType,iZoneID,iLUType,iZExtent,    &
```

```
                                iNZones,iElems(iNZones),iLayers(iNZones), &
                                iZoneIDs(iNZones),iLenDate,iNFlows_In,    &
                                iNTimes_In,iLenFlowNames
    CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),cEndDate(iLenDate)
    REAL(C_DOUBLE),INTENT(IN) :: rFactVL
    REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNFlows_In,iNTimes_In)
    CHARACTER(C_CHAR),INTENT(OUT) :: cFlowNames(iLenFlowNames)
    INTEGER(C_INT),INTENT(OUT) :: iNFlows_Out,iNTimes_Out,         &
                                  iLocArray(iNFlows_In),           &
                                  iWaterYears(iNTimes_In),iStat
END SUBROUTINE IW_Model_GetZBudget_AnnualFlows
```

`iZBudgetType` : Z-Budget type for the selected Z-Budget file; available Z-Budget types for the model can be retrieved by calling procedure `IW_Model_GetZBudget_List` (see section 8.1.93)

`iZoneID` : Identification number of the selected zone for which monthly average water budget flows from the selected Z-Budget file is being retrieved

`iLUType` : Land use type identification number for which annual water budget flows are being retrieved (note that some Z-Budget files such as groundwater Z-Budget will return the same data regardless of the value of this argument); the land use identification numbers can be retrieved by calling the `IW_GetLandUseTypeIDs` procedure from the *Miscellaneous* procedures group (see section 8.4.11)

`iZExtent` : Extent of the zone definition (horizontal or vertical; zone definition codes can be retrieved by calling `IW_GetZoneExtentID_***` procedures from the *Miscellaneous* procedures group)

`iNZones` : Number of model cells for which a zone number is defined

`iElems` : Array of element numbers for which a zone number is specified

`iLayers` : Array of model layers where elements listed in the `iElems` argument reside; if `iZExtent` is set for horizontal zone definition, components of this array can all be set to 1

`iZoneIDs` : Array of zone numbers defined for the elements listed in the `iElems` array

`iLenDate` : Character length of the beginning and ending dates of the simulation period for which the monthly average water terms will be retrieved; this argument should be set to a minimum of 16

`cBeginDate` : Beginning date, in IWFM timestamp format, of the simulation period for which water budget data is being retrieved; this should be an October date

`cEndDate` : Ending date, in IWFM timestamp format, of the simulation period for which water budget data is being retrieved; this should be a September date

`rFactVL` : Factor to convert simulation unit of flow to desired unit of flow

`iNFlows_In` : Maximum number of water budget flow components that will be retrieved; this can be set to the number of columns in the Z-Budget file obtained by calling procedure `IW_Model_GetZBudget_NColumns` (see section 8.1.94)

`iNTimes_In` : Maximum number of years for which data will be retrieved; this can be set to the number of timesteps in the simulation period which can be obtained by calling procedure `IW_Model_GetNTimeSteps` (see section 8.1.4)

`rFlows` : 2-dimensional array that will be returned with the annual water budget flows; the first dimension of the array is used for the water budget column and the second dimension is used for each water year within the data retrieval period defined by `cBeginDate` and `cEndDate`

`iNFlows_Out` : Actual number of flow terms involved in the water budget; this number excludes columns in the Z-Budget file used as informational columns (e.g. area of non-ponded crops in Land & Water Use Z-Budget file)

iNTimes_Out	:	Actual number of water years for which data was retrieved
iLenFlowNames	:	Character length of the description of water budget flow terms concatenated into a string scalar; a value of 3000 is appropriate
cFlowNames	:	Description of water budget flow terms concatenated into a string scalar
iLocArray	:	Character position array so that client software can separate the cFlowNames scalar string argument into an array of water budget flow term descriptions
iWaterYears	:	List of water years for which data was retrieved
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.98. IW_Model_GetZBudget_MonthlyFlows

This procedure retrieves the monthly water budget flows from a specified Z-Budget output file for a specified zone along with the description of the flow terms. It should be noted that this procedure does not retrieve all the columns stored in the Z-Budget file, only those that are an active inflow or outflow component of a water budget (some columns such as area of rice fields in the Land & Water Use Z-Budget represent informational data that are not part of the water budget).

```
SUBROUTINE IW_Model_GetZBudget_MonthlyFlows(iZBudgetType,iZoneID,         &
              iLUType,iZExtent,iNZones,iElems,iLayers,iZoneIDs,iLenDate,  &
              cBeginDate,cEndDate,rFactVL,iNFlows_In,iNTimes_In,rFlows,   &
              iNFlows_Out,iNTimes_Out,iLenFlowNames,cFlowNames,iLocArray, &
              iStat)
    INTEGER(C_INT),INTENT(IN) :: iZBudgetType,iZoneID,iLUType,iZExtent,    &
                                 iNZones,iElems(iNZones),iLayers(iNZones), &
                                 iZoneIDs(iNZones),iLenDate,iNFlows_In,    &
                                 iNTimes_In,iLenFlowNames
    CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),cEndDate(iLenDate)
    REAL(C_DOUBLE),INTENT(IN) :: rFactVL
    REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNFlows_In,iNTimes_In)
    CHARACTER(C_CHAR),INTENT(OUT) :: cFlowNames(iLenFlowNames)
    INTEGER(C_INT),INTENT(OUT) :: iNFlows_Out,iNTimes_Out,         &
                                  iLocArray(iNFlows_In),iStat
END SUBROUTINE IW_Model_GetZBudget_MonthlyFlows
```

`iZBudgetType` : Z-Budget type for the selected Z-Budget file; available Z-Budget types for the model can be retrieved by calling procedure `IW_Model_GetZBudget_List` (see section 8.1.93)

`iZoneID` : Identification number of the selected zone for which monthly average water budget flows from the selected Z-Budget file is being retrieved

`iLUType` : Land use type identification number for which monthly water budget flows are being retrieved (note that some Z-Budget files such as groundwater Z-Budget will return the same data regardless of the value of this argument); the land use identification numbers can be retrieved by calling the `IW_GetLandUseTypeIDs` procedure from the *Miscellaneous* procedures group (see section 8.4.11)

`iZExtent` : Extent of the zone definition (horizontal or vertical; zone definition codes can be retrieved by calling `IW_GetZoneExtentID_***` procedures from the *Miscellaneous* procedures group)

`iNZones` : Number of model cells for which a zone number is defined

`iElems` : Array of element numbers for which a zone number is specified

`iLayers` : Array of model layers where elements listed in the `iElems` argument reside; if `iZExtent` is set for horizontal zone definition, components of this array can all be set to 1

`iZoneIDs` : Array of zone numbers defined for the elements listed in the `iElems` array

`iLenDate` : Character length of the beginning and ending dates of the simulation period for which the monthly average water terms will be retrieved; this argument should be set to a minimum of 16

`cBeginDate` : Beginning date, in IWFM timestamp format, of the simulation period for which water budget data is being retrieved; this should be an October date

`cEndDate` : Ending date, in IWFM timestamp format, of the simulation period for which water budget data is being retrieved; this should be a September date

`rFactVL` : Factor to convert simulation unit of flow to desired unit of flow

`iNFlows_In` : Maximum number of water budget flow components that will be retrieved; this can be set to the number of columns in the Z-Budget file obtained by calling procedure `IW_Model_GetZBudget_NColumns` (see section 8.1.94)

`iNTimes_In` : Maximum number of months for which data will be retrieved; this can be set to the number of timesteps in the simulation period which can be obtained by calling procedure `IW_Model_GetNTimeSteps` (see section 8.1.4)

`rFlows` : 2-dimensional array that will be returned with the monthly water budget flows; the first dimension of the array is used for the water budget column and the second dimension is used for each month within the data retrieval period defined by `cBeginDate` and `cEndDate`

`iNFlows_Out` : Actual number of flow terms involved in the water budget; this number excludes columns in the Z-Budget file used as informational columns (e.g. area of non-ponded crops in Land & Water Use Z-Budget file)

`iNTimes_Out` : Actual number of months for which data was retrieved

`iLenFlowNames` : Character length of the description of water budget flow terms concatenated into a string scalar; a value of 3000 is appropriate

`cFlowNames` : Description of water budget flow terms concatenated into a string scalar

`iLocArray` : Character position array so that client software can separate the `cFlowNames` scalar string argument into an array of water budget flow term descriptions

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.99. IW_Model_GetZBudget_MonthlyAverageFlows

This procedure retrieves the average monthly water budget flows from a specified Z-Budget output file for a specified zone along with the description of the flow terms. It should be noted that this procedure does not retrieve the average of all the columns stored in the Z-Budget file, only those that are an active inflow or outflow component of a water budget (some columns such as area of rice fields in the Land & Water Use Z-Budget represent informational data that are not part of the water budget).

```
SUBROUTINE IW_Model_GetZBudget_MonthlyAverageFlows(iZBudgetType,iZoneID,   &
             iLUType,iZExtent,iNZones,iElems,iLayers,iZoneIDs,iLenDate,   &
             cBeginDate,cEndDate,rFactVL,iNFlows_In,rFlows,iNFlows_Out,   &
             iLenFlowNames,cFlowNames,iLocArray,iStat)
   INTEGER(C_INT),INTENT(IN) :: iZBudgetType,iZoneID,iLUType,       &
                                iZExtent,iNZones,iElems(iNZones),   &
                                iLayers(iNZones),iZoneIDs(iNZones), &
                                iLenDate,iNFlows_In,iLenFlowNames
   CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),cEndDate(iLenDate)
   REAL(C_DOUBLE),INTENT(IN) :: rFactVL
   REAL(C_DOUBLE),INTENT(OUT) :: rFlows(iNFlows_In,12)
   CHARACTER(C_CHAR),INTENT(OUT) :: cFlowNames(iLenFlowNames)
   INTEGER(C_INT),INTENT(OUT) :: iNFlows_Out,iLocArray(iNFlows_In),  &
                                 iStat
END SUBROUTINE IW_Model_GetZBudget_MonthlyAverageFlows
```

`iZBudgetType` : Z-Budget type for the selected Z-Budget file; available Z-Budget types for the model can be retrieved by calling procedure `IW_Model_GetZBudget_List` (see section 8.1.93)

`iZoneID` : Identification number of the selected zone for which monthly average water budget flows from the selected Z-Budget file is being retrieved

`iLUType` : Land use type identification number for which average monthly water budget flows are being retrieved (note that some Z-Budget files such as groundwater Z-Budget will return the same data regardless of the value of this argument); the land use identification numbers can be retrieved by calling the `IW_GetLandUseTypeIDs` procedure from the *Miscellaneous* procedures group (see section 8.4.11)

`iZExtent` : Extent of the zone definition (horizontal or vertical; zone definition codes can be retrieved by calling `IW_GetZoneExtentID_***` procedures from the *Miscellaneous* procedures group)

`iNZones` : Number of model cells for which a zone number is defined

`iElems` : Array of element numbers for which a zone number is specified

`iLayers` : Array of model layers where elements listed in the `iElems` argument reside; if `iZExtent` is set for horizontal zone definition, components of this array can all be set to 1

`iZoneIDs` : Array of zone numbers defined for the elements listed in the `iElems` array

`iLenDate` : Character length of the beginning and ending dates of the simulation period for which the monthly average water terms will be retrieved; this argument should be set to a minimum of 16

`cBeginDate` : Beginning date, in IWFM timestamp format, of the simulation period for which water budget data is being retrieved; this should be an October date

`cEndDate` : Ending date, in IWFM timestamp format, of the simulation period for which water budget data is being retrieved; this should be a September date

`rFactVL` : Factor to convert simulation unit of flow to desired unit of flow

`iNFlows_In` : Maximum number of water budget flow components that will be retrieved; this can be set to the number of columns in the Z-Budget file obtained by calling procedure `IW_Model_GetZBudget_NColumns` (see section 8.1.94)

`rFlows` : 2-dimensional array that will be returned with the monthly average water budget flows; the first dimension of the array is used for the water budget column and the second dimension is used for each month starting with October and ending with September

`iNFlows_Out` : Actual number of flow terms involved in the water budget; this number excludes columns in the Z-Budget file used as informational columns (e.g. area of non-ponded crops in Land & Water Use Z-Budget file)

`iLenFlowNames` : Character length of the description of water budget flow terms concatenated into a string scalar; a value of 3000 is appropriate

`cFlowNames` : Description of water budget flow terms concatenated into a string scalar

`iLocArray` : Character position array so that client software can separate the `cFlowNames` scalar string argument into an array of water budget flow term descriptions

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.100. IW_Model_GetZBudget_CumGWStorChange

This procedure retrieves the cumulative change in groundwater storage from the Groundwater Z-Budget file for a selected zone with a specified time interval for a specified time period. The client can provide unit conversion factors to convert the retrieved data from the simulation units to a different unit.

```
SUBROUTINE IW_Model_GetZBudget_CumGWStorChange(iZoneID,iZExtent,iNZones,   &
             iElems,iLayers,iZoneIDs,iLenDate,cBeginDate,cEndDate,         &
             iLenInterval,cInterval,rFactVL,rOutputDates,iNTimes_In,       &
             rCumGWStorChange,iNTimes_Out,iStat)
   INTEGER(C_INT),INTENT(IN) ::iZoneID,iZExtent,iNZones,iElems(iNZones),   &
                         iLayers(iNZones),iZoneIDs(iNZones),iLenDate,      &
                         iLenInterval,iNTimes_In
   CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),                   &
                         cEndDate(iLenDate),cInterval(iLenInterval)
   REAL(C_DOUBLE),INTENT(IN) :: rFactVL
   REAL(C_DOUBLE),INTENT(OUT) :: rOutputDates(iNTimes_In),                 &
                                 rCumGWStorChange(iNTimes_In)
   INTEGER(C_INT),INTENT(OUT) :: iNTimes_Out,iStat
END SUBROUTINE IW_Model_GetZBudget_CumGWStorChange
```

`iZoneID` : Identification number of the selected zone for which cumulative change in groundwater storage will be retrieved

`iZExtent` : Extent of the zone definition (horizontal or vertical; zone definition codes can be retrieved by calling `IW_GetZoneExtentID_***` procedures from the *Miscellaneous* procedures group)

`iNZones` : Number of model cells for which a zone number is defined

`iElems` : Array of element numbers for which a zone number is specified

`iLayers` : Array of model layers where elements listed in the `iElems` argument reside; if `iZExtent` is set for horizontal zone definition, components of this array can all be set to 1

`iZoneIDs` : Array of zone numbers defined for the elements listed in the `iElems` array

`iLenDate` : Character length of the beginning and ending dates of the simulation period for which the cumulative change in groundwater storage will be retrieved; this argument should be set to a minimum of 16

`cBeginDate` : Beginning date for data retrieval in IWFM timestamp format

`cEndDate` : Ending date for data retrieval in IWFM timestamp format

`iLenInterval` : Character length of the time interval at which timeseries data will be retrieved

`cInterval` : Time interval of the retrieved data (e.g. retrieving the data with monthly interval when a model was run with a daily timestep); a list of possible time intervals for data retrieval can be obtained by calling procedure `IW_Model_GetOutputIntervals` (see section 8.1.6)

`rFactVL` : Factor to convert simulation unit of groundwater storage to desired unit of storage

`rOutputDates` : Array of dates corresponding to each retrieved data point given in Julian format; Julian day 1 in this format corresponds to January 1, 1900 to comply with MS Excel Julian format

`iNTimes_In` : Maximum number of timesteps for which data will be retrieved; this can be set to the number of timesteps in the simulation period which can be obtained by calling procedure `IW_Model_GetNTimeSteps` (see section 8.1.4)

`rCumGWStorChange` : Array that will be returned with the retrieved cumulative change in groundwater storage data

`iNTimes_Out` : Actual number of timesteps for which data was retrieved

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.101. IW_Model_GetZBudget_AnnualCumGWStorChange

This procedure retrieves annual cumulative change in groundwater storage from the Groundwater Z-Budget file for a selected zone for a specified time period. Annual values represent each water year that starts on October 1st of a year and ends on September 30th of the following year. This procedure adjusts, if needed, the beginning and ending of the data retrieval period specified by the client so that the data can be properly compiled for water years. The client can provide unit conversion factors to convert the retrieved data from the simulation units to a different unit.

```
SUBROUTINE IW_Model_GetZBudget_AnnualCumGWStorChange(iZoneID,iZExtent,     &
              iNZones,iElems,iLayers,iZoneIDs,iLenDate,cBeginDate,         &
              cEndDate,rFactVL,iNTimes_In,rCumGWStorChange,iWaterYears,    &
              iNTimes_Out,iStat)
    INTEGER(C_INT),INTENT(IN) ::iZoneID,iZExtent,iNZones,iElems(iNZones),  &
                         iLayers(iNZones),iZoneIDs(iNZones),iLenDate,     &
                         iNTimes_In
    CHARACTER(C_CHAR),INTENT(IN) :: cBeginDate(iLenDate),cEndDate(iLenDate)
    REAL(C_DOUBLE),INTENT(IN) :: rFactVL
    REAL(C_DOUBLE),INTENT(OUT) :: rCumGWStorChange(iNTimes_In)
    INTEGER(C_INT),INTENT(OUT) :: iWaterYears(iNTimes_In),iNTimes_Out,iStat
END SUBROUTINE IW_Model_GetZBudget_AnnualCumGWStorChange
```

`iZoneID` : Identification number of the selected zone for which annual cumulative change in groundwater storage will be retrieved

`iZExtent` : Extent of the zone definition (horizontal or vertical; zone definition codes can be retrieved by calling `IW_GetZoneExtentID_***` procedures from the *Miscellaneous* procedures group)

`iNZones` : Number of model cells for which a zone number is defined

`iElems` : Array of element numbers for which a zone number is specified

`iLayers` : Array of model layers where elements listed in the `iElems` argument reside; if `iZExtent` is set for horizontal zone definition, components of this array can all be set to 1

`iZoneIDs`	:	Array of zone numbers defined for the elements listed in the `iElems` array
`iLenDate`	:	Character length of the beginning and ending dates of the simulation period for which the annual cumulative change in groundwater storage will be retrieved; this argument should be set to a minimum of 16
`cBeginDate`	:	Beginning date for data retrieval in IWFM timestamp format
`cEndDate`	:	Ending date for data retrieval in IWFM timestamp format
`rFactVL`	:	Factor to convert simulation unit of groundwater storage to desired unit of storage
`iNTimes_In`	:	Maximum number of timesteps for which data will be retrieved; this can be set to the number of timesteps in the simulation period which can be obtained by calling procedure `IW_Model_GetNTimeSteps` (see section 8.1.4)
`rCumGWStorChange`	:	Array that will be returned with the retrieved annual cumulative change in groundwater storage data
`iWaterYears`	:	List of water years for which data was retrieved
`iNTimes_Out`	:	Actual number of timesteps for which data was retrieved
`iStat`	:	Error code; returns 0 if the procedure call was successful

8.1.102. IW_Model_GetLocationTypeIDs_WithAvailableData

This procedure retrieves a list of location type identification codes for which data is available for retrieval for post-processing purposes.

```
SUBROUTINE IW_Model_GetLocationTypeIDs_WithAvailableData(iMaxDim,  &
              iLocationTypeIDs,iActualDim,iStat)
   INTEGER(C_INT),INTENT(IN) :: iMaxDim
   INTEGER(C_INT),INTENT(OUT) :: iLocationTypeIDs(iMaxDim), iActualDim,  &
                                 iStat
END SUBROUTINE IW_Model_GetLocationTypeIDs_WithAvailableData
```

`iMaxDim` : Maximum number of location type codes that can have associated data for retrieval; set this to argument to a minimum value of 15

`iLocationTypeIDs` : List of codes of location types for which model data can be retrieved for post-processing purposes; all possible location type codes can be obtained by calling the `IW_GetLocationTypeIDs` procedure (see section 8.4.24) from the *Miscellaneous* procedures group

`iActualDim` : Number of location types for which model data can be retrieved

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.103. IW_Model_GetNDataList_AtLocationType

This procedure retrieves the number of data types available for retrieval at a selected location type. Location type codes can be obtained by calling the `IW_GetLocationTypeIDs` procedure (see section 8.4.24) from the *Miscellaneous* procedures group.

```
SUBROUTINE IW_Model_GetNDataList_AtLocationType(iLocationType,nData,iStat)
    INTEGER(C_INT),INTENT(IN) :: iLocationType
    INTEGER(C_INT),INTENT(OUT) :: nData,iStat
END SUBROUTINE IW_Model_GetNDataList_AtLocationType
```

`iLocationType` : Location type code which can be obtained by calling the `IW_GetLocationTypeIDs` procedure (see section 8.4.24) from the *Miscellaneous* procedures group

`nData` : Number of available data types for retrieval at the selected location type, `iLocationType`

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.104. IW_Model_GetDataList_AtLocationType

This procedure retrieves data types available for retrieval at a selected location type. Location type codes can be obtained by calling the `IW_GetLocationTypeIDs` procedure (see section 8.4.24) from the *Miscellaneous* procedures group.

```
SUBROUTINE IW_Model_GetDataList_AtLocationType(iLocationType,nData,   &
             iDimLocArray,iLocArray,iLenDataList,cDataListSend,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLocationType,iDimLocArray,iLenDataList
   INTEGER(C_INT),INTENT(OUT) :: iLocArray(iDimLocArray),nData,iStat
   CHARACTER(C_CHAR),INTENT(OUT) :: cDataListSend(iLenDataList)
END SUBROUTINE IW_Model_GetDataList_AtLocationType
```

`iLocationType` : Location type code which can be obtained by calling the `IW_GetLocationTypeIDs` procedure (see section 8.4.24) from the *Miscellaneous* procedures group

`nData` : Number of actual types of data that is available at the selected location type, `iLocationType`

`iDimLocArray` : Maximum number of data types that can be retrieved at a selected location type, `iLocationType`; a value of 20 is appropriate

`iLocArray` : Character position array so that client software can separate the `cDataListSend` scalar string argument into individual data types

`iLenDataList` : Character length of the `cDataListSend` argument which must be set to a large enough value to store all data types; a value of 3000 is appropriate

`cDataListSend` : List of data types that can be retrieved for the selected location type, `iLocationType`, concatenated into a scalar string argument

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.105. IW_Model_GetSubDataList_ForLocationAndDataType

This procedure retrieves the list of available sub-data types for selected data and location types. Not all data types have sub-data types.

```
SUBROUTINE IW_Model_GetSubDataList_ForLocationAndDataType(iLocationType,  &
             iSelectedDataIndex,nData,iDimLocArray,iLocArray,              &
             iLenDataList,cDataListSend,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLocationType,iSelectedDataIndex,          &
                                iDimLocArray,iLenDataList
   INTEGER(C_INT),INTENT(OUT) :: iLocArray(iDimLocArray),nData,iStat
   CHARACTER(C_CHAR),INTENT(OUT) :: cDataListSend(iLenDataList)
END SUBROUTINE IW_Model_GetSubDataList_AtLocation
```

`iLocationType` : Location type code which can be obtained by calling the `IW_GetLocationTypeIDs` procedure from the *Miscellaneous* procedures group

`iSelectedDataIndex`: Index of the data type selected by the client; the index is based on the order of data types that was retrieved by calling procedure `IW_Model_GetDataList_AtLocationType` (see section 8.1.104)

`nData` : Number of actual types of sub-data that is available at the selected location type, `iLocationType`; if the selected data type does not have sub-data, `nData` will be returned as 0 (zero)

`iDimLocArray` : Maximum number of sub-data types that can be retrieved at a selected location type, `iLocationType`, and identification number, `iLocationID`; a value of 100 is appropriate

`iLocArray` : Character position array so that client software can separate the `cDataListSend` scalar string argument into individual sub-data types

`iLenDataList` : Character length of the `cDataListSend` argument which must be set to a large enough value to store all data types; a value of 3000 is appropriate

`cDataListSend` : List of sub-data types that can be retrieved for the selected location type, `iLocationType`, with the location identification

number, iLocationID, concatenated into a scalar string argument

iStat : Error code; returns 0 if the procedure call was successful

8.1.106. IW_Model_GetModelData_AtLocation

This procedure retrieves time-series simulation data at a selected location with a specified time interval for a specified time period. The client can provide unit conversion factors to convert the retrieved time-series data from the simulation units to a different unit. The procedure also retrieves information about the type of the time-series data units; i.e. if the retrieved time-series data is length type, area type or volume type. Data unit type codes used by IWFM API can be obtained by calling the IW_GetDataUnitTypeIDs procedure from the *Miscellaneous* procedures group.

```
SUBROUTINE IW_Model_GetModelData_AtLocation(iLocationType,iLocation,       &
             iDataTypeIndex,iSubDataIndex,iZExtent,iDimZones,iElems,       &
             iLayers,iZones,cOutputBeginDateAndTime,                       &
             cOutputEndDateAndTime,iLenOutputDateAndTime,cOutputInterval,  &
             iLenOutputInterval,rFact_LT,rFact_AR,rFact_VL,iDataUnitType,  &
             iDimOutput_In,iDimOutput_Out,rOutputDates,                    &
             rOutputValues,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLocationType,iLocation,iDataTypeIndex,    &
                         iSubDataIndex,iZExtent,iDimZones,                 &
                         iElems(iDimZones),iLayers(iDimZones),             &
                         iZones(iDimZones),iLenOutputDateAndTime,          &
                         iLenOutputInterval,iDimOutput_In
   CHARACTER(C_CHAR),INTENT(IN) ::                                         &
                         cOutputBeginDateAndTime(iLenOutputDateAndTime),   &
                         cOutputEndDateAndTime(iLenOutputDateAndTime)      &
                         cOutputInterval(iLenOutputInterval)
   REAL(C_DOUBLE),INTENT(IN) :: rFact_LT,rFact_AR,rFact_VL
   INTEGER(C_INT),INTENT(OUT) :: iDataUnitType,iDimOutput_Out,iStat
   REAL(C_DOUBLE),INTENT(OUT) :: rOutputDates(iDimOutput_In),              &
                                 rOutputValues(iDimOutput_In)
END SUBROUTINE IW_Model_GetModelData_AtLocation
```

iLocationType : Location type code which can be obtained by calling the IW_GetLocationTypeIDs procedure from the *Miscellaneous* procedures group

`iLocation` : Location index; e.g. subregion index if location type is subregion, index of finite element cell if location type is finite element, lake index if location type is lake

`iDataTypeIndex` : Index of the data type selected by the client; the index is based on the order of data types that was retrieved by calling `IW_Model_GetDataList_AtLocationType` (see section 8.1.104) procedure

`iSubDataIndex` : Index of the sub-data type selected by the client; the index is based on the order of sub-data types that was retrieved by calling procedure `IW_Model_GetSubDataList_ForLocationAndDataType` (see section 8.1.105

`iZExtent` : Extent of the zone definition (horizontal or vertical; zone definition codes can be retrieved by calling `IW_GetZoneExtentID_***` procedures from the *Miscellaneous* procedures group); this information is only used by IWFM API when location type (see `iLocationType` argument above) is a zone

`iDimZones` : Number of model cells for which a zone number is defined; IWFM API uses this information only if the location type (see `iLocationType` argument above) is a zone, otherwise it can be set as zero

`iElems` : Array of element numbers for which a zone number is specified; IWFM API uses this information only if the location type (see `iLocationType` argument above) is a zone

`iLayers` : Array of model layers where elements listed in the `iElems` argument reside; IWFM API uses this information only if the location type (see `iLocationType` argument above) is a zone (if `iZExtent` is set for horizontal zone definition, components of this array can all be set to 1)

`iZones` : Array of zone numbers defined for the elements listed in the `iElems` array; IWFM API uses this information only if the location type (see `iLocationType` argument above) is a zone

`cOutputBeginDateAndTime` : Time-series data retrieval beginning date and time specified in IWFM timestamp format

`cOutputEndDateAndTime` : Time-series data retrieval ending date and time specified in IWFM timestamp format

`iLenOutputDateAndTime` : Character length of `cOutputBeginDateAndTime` and `cOutputEndDateAndTime` arguments

`cOutputInterval` : Time interval of the retrieved time-series data; e.g. retrieval groundwater pumping values with monthly interval when a model was run with a daily timestep

`iLenOutputInterval` : Character length of `cOutputInterval` argument

`rFact_LT` : Conversion factor to convert length-type time-series data from simulation unit of length to a desired unit of length

`rFact_AR` : Conversion factor to convert area-type time-series data from simulation unit of area to a desired unit of area

`rFact_VL` : Conversion factor to convert volume-type time-series data from simulation unit of volume to a desired unit of volume

`iDataUnitType` : Data unit type (length, area or volume) that is being returned; data unit type codes used by the IWFM API can be retrieved using `IW_GetDataUnitTypeIDs` procedure from the *Miscellaneous* procedures group

`iDimOutput_In` : Maximum number of data points that will be retrieved; this value is the number of simulation timesteps and can be calculated using the `IW_GetNIntervals` procedure

(see section 8.4.42) from the *Miscelleneous* procedures group

`iDimOutput_Out` : Actual number of data points that is retrieved

`rOutputDates` : Array of dates corresponding to each retrieved data point given in Julian format; Julian day 1 in this format corresponds to January 1, 1900 to comply with MS Excel Julian format

`rOutputValues` : Retrieved time-series data points

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.107. IW_Model_GetModelData_GWHeadsAll_ForALayer

This procedure retrieves groundwater heads at all nodes in a specified aquifer layer for a specified time period. It is intended to be used after an IWFM model is run to completion and the groundwater heads at all nodes and layers are printed to an output file.

```
SUBROUTINE IW_Model_GetModelData_GetGWHeadsAll_ForALayer(iLayer,..   &
              cOutputBeginDateAndTime,cOutputEndDateAndTime,         &
              iLenDateAndTime,rFact_LT,iNNodes, iNTime,rOutputDates, &
              iNTime,rOutputDates,rGWHeads,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLayer,iNNodes,iNTime,iLenDateAndTime
   REAL(C_DOUBLE),INTENT(IN) :: rFact_LT
   REAL(C_DOUBLE),INTENT(OUT) :: rOutputDates(iNTime),   &
                                 rGWHeads(iNNodes,iNTime)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetModelData_GetGWHeadsAll_ForALayer
```

`iLayer` : Aquifer layer number for which gw heads at all nodes are being retrieved

`cOutputBeginDateAndTime` : Groundwater heads retrieval beginning date and time specified in IWFM timestamp format

`cOutputEndDateAndTime` : groundwater heads retrieval ending date and time specified in IWFM timestamp format

iLenDateAndTime	:	Character length of cOutputBeginDateAndTime and cOutputEndDateAndTime arguments
rFact_LT	:	Conversion factor to convert groundwater heads from simulation unit of length to a desired unit of length
iNNodes	:	Number of grid nodes; this information can be obtained by calling IW_Model_GetNNodes (see section 8.1.6)
iNTime	:	Number of simulation timesteps between cOutputBeginDateAndTime and cOutputEndDateAndTime; this information can be obtained by calling IW_GetNIntervals method (see section 8.4.42)
rOutputDates	:	Array of dates corresponding to each set of retrieved groundwater heads given in Julian format; Julian day 1 in this format corresponds to January 1, 1900 to comply with MS Excel Julian format
rGWHeads	:	Retrieved groundwater heads for all nodes at the specified layer for the specified period
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.108. IW_Model_GetGWHeads_All

This procedure retrieves groundwater heads at all nodes in every aquifer layer. It is intended to be used while an IWFM model is running; e.g. to retrieve groundwater heads after one timestep is simulated using the IW_Model_SimulateForOneTimeStep (see section 8.1.118) method. To retrieve groundwater heads at all nodes and layers after an IWFM model is run to completion, use IW_Model_GetModelData_GWHeadsAll_ForALayer (see section 8.1.107) method.

```
SUBROUTINE IW_Model_GetGWHeads_All(iNNodes,iNLayers,iPrevious,rFact,  &
             Heads,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNNodes,iNLayers,iPrevious
   REAL(C_DOUBLE),INTENT(IN) :: rFact
   REAL(C_DOUBLE),INTENT(OUT) :: Heads(iNNodes,iNLayers)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetGWHeads_All
```

iNNodes	:	Number of grid nodes
iNLayers	:	Number of aquifer layers
iPrevious	:	Flag to specify if the groundwater heads at the beginning or at the end of the timestep are being retrieved; 0 = heads at the end of the timestep are being retrieved, 1= heads at the beginning of timestep are being retrieved
rFact	:	Conversion factor to convert groundwater heads from simulation unit of length to a desired unit of length
Heads	:	Groundwater heads at each node and layer
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.109. IW_Model_GetSubsidence_All

This procedure retrieves subsidence at all nodes in every aquifer layer. It is intended to be used while an IWFM model is running; e.g. to retrieve groundwater heads after one timestep is simulated using the IW_Model_SimulateForOneTimeStep (see section 8.1.118) method.

```
SUBROUTINE IW_Model_GetSubsidence_All(iNNodes,iNLayers,rFact,  &
             Subs,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNNodes,iNLayers
    REAL(C_DOUBLE),INTENT(IN) :: rFact
    REAL(C_DOUBLE),INTENT(OUT) :: Subs(iNNodes,iNLayers)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetSubsidence_All
```

iNNodes	:	Number of grid nodes
iNLayers	:	Number of aquifer layers
rFact	:	Conversion factor to convert subsidence from simulation unit of length to a desired unit of length
Subs	:	Subsidence at each node and layer
iStat	:	Error code; returns 0 if the procedure call was successful

8.1.110. IW_Model_GetSubregionAgPumpingAverageDepthToGW

This procedure retrieves subregional depth-to-groundwater values that are weighted-averaged with respect to agricultural pumping rates. This procedure was initially developed to link IWFM model applications to agricultural economics models which need depth-to-groundwater as a surrogate to cost of pumping.

```
SUBROUTINE IW_Model_GetSubregionAgPumpingAverageDepthToGW(NSubregions,  &
             AveDepthToGW,iStat)
   INTEGER(C_INT),INTENT(IN)  :: NSubregions
   REAL(C_DOUBLE),INTENT(OUT) :: AveDepthToGW(NSubregions)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_GetSubregionAgPumpingAverageDepthToGW
```

`NSubregions` : Number of subregions; this value can be obtained by calling procedure `IW_Model_GetNSubregions` (see section 8.1.13)

`AveDepthToGW` : Depth-to-groundwater at each subregion weighted-averaged with respect to agricultural pumping rates

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.111. IW_Model_GetNLocations

This procedure retrieves the number of locations (e.g. subregions, lakes, nodes, elements, etc.) for a specified location type. This is a generic version of procedures such as `IW_Model_GetNElements`, `IW_Model_GetNNodes`, `IW_Model_GetNSubregions`, etc.

```
SUBROUTINE IW_Model_GetNLocations(iLocType,iNLocations,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLocationType
   INTEGER(C_INT),INTENT(OUT) :: iNLocations,iStat
END SUBROUTINE IW_Model_GetNLocations
```

`iLocType` : Location type code which can be obtained by calling the `IW_GetLocationTypeIDs` procedure from the *Miscellaneous* procedures group (see section 8.4.24)

`iNLocations` : Number of locations for the selected location type, `iLocationType`

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.112. IW_Model_GetLocationIDs

This procedure retrieves the location (e.g. subregions, lakes, nodes, elements, etc.) identification numbers used by the IWFM model for a specified location type. This is a generic version of procedures such as `IW_Model_GetElementIDs`, `IW_Model_GetNodeIDs`, `IW_Model_GetSubregionIDs`, etc.

```
SUBROUTINE IW_Model_GetLocationIDs(iLocType,iNLocations,iLocationIDs,iStat)
    INTEGER(C_INT),INTENT(IN) :: iLocType,iNLocations
    INTEGER(C_INT),INTENT(OUT) :: iLocationIDs(iNLocations),iStat
END SUBROUTINE IW_Model_GetLocationIDs
```

`iLocType` : Location type code which can be obtained by calling the `IW_GetLocationTypeIDs` procedure from the *Miscellaneous* procedures group (see section 8.4.24)

`iNLocations` : Number of locations for the selected location type, `iLocationType`; it should have been obtained by calling procedure `IW_Model_GetNLocations` (see section 8.1.111)

`iLocationIDs` : Location identification numbers used by the IWFM model

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.113. IW_Model_SetPreProcessorPath

This procedure defines the path to the directory where the *Preprocessor Main Input File* is located.

```
SUBROUTINE IW_Model_SetPreProcessorPath(iLen,cPath,iStat)
    INTEGER(C_INT),INTENT(IN) :: iLen
    CHARACTER(C_CHAR),INTENT(IN) :: cPath(iLen)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_SetPreProcessorPath
```

`iLen` : Character length of the path for the *Preprocessor Main Input File*

`cPath` : Path for the Preprocessor Main Input File

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.114. IW_Model_SetSimulationPath

This procedure defines the path to the directory where the *Simulation Main Input File* is located.

```
SUBROUTINE IW_Model_SetSimulationPath(iLen,cPath,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLen
   CHARACTER(C_CHAR),INTENT(IN) :: cPath(iLen)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_SetSimulationPath
```

`iLen` : Character length of the path for the *Simulation Main Input File*

`cPath` : Path for the Simulation Main Input File

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.115. IW_Model_SetSupplyAdjustmentMaxIters

This procedure sets the maximum number of iterations that will be used in automatic supply adjustment.

```
SUBROUTINE IW_Model_SetSupplyAdjustmentMaxIters(iMaxIters,iStat)
   INTEGER(C_INT),INTENT(IN) :: iMaxIters
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_SetSupplyAdjustmentMaxIters
```

`iMaxIters` : Maximum number of iterations for automatic supply adjustment

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.116. IW_Model_SetSupplyAdjustmentTolerance

This procedure sets the tolerance, given as a fraction of the water demand (e.g. 0.01 represents 1% of the demand) that will be used in automatic supply adjustment. When the automatic supply adjustment feature of IWFM is turned on, IWFM iteratively tries to adjust water supplies (diversions, pumping or both based on user-

defined specifications) to meet the water demand. When the difference between water supply and demand is less than the tolerance, IWFM assumes equivalency between demand and supply, and terminates supply adjustment iterations.

```
SUBROUTINE IW_Model_SetSupplyAdjustmentTolerance(rToler,iStat)
    REAL(C_DOUBLE),INTENT(IN) :: rToler
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_SetSupplyAdjustmentTolerance
```

`rToler` : Fraction of water demand to be used as convergence criteria for iterative supply adjustment

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.117. IW_Model_DeleteInquiryDataFile

When a Model object is instantiated for inquiry purposes (i.e. by setting the `iIsForInquiry` = 1 in the `IW_Model_New` procedure), the API generates a binary file, *IW_ModelData_ForInquiry.bin*, to store the model related summary data. This file is used in the future to quickly instantiate the Model object. When the IWFM model application parameters are changed (for example when a scenario run is performed), this binary file must be deleted to allow the generation of a new *IW_ModelData_ForInquiry.bin* file that stores information related to the new model set-up. This procedure is used to delete this binary inquiry.

```
SUBROUTINE IW_Model_DeleteInquiryDataFile(iLenSimFileName,cSimFileName,  &
              iStat)
    INTEGER(C_INT),INTENT(IN) :: iLenSimFileName
    CHARACTER(C_CHAR),INTENT(IN) :: cSimFileName(iLenSimFileName)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_DeleteInquiryDataFile
```

`iLenSimFileName` : Character length of the path for the *Simulation Main Input File*

`cPath` : Path for the Simulation Main Input File

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.118. IW_Model_SimulateForOneTimeStep

This procedure performs hydrologic simulations for a single timestep of the IWFM model application.

```
SUBROUTINE IW_Model_SimulateForOneTimeStep(iStat)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_SimulateForOneTimeStep
```

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.119. IW_Model_SimulateForAnInterval

This procedure performs hydrologic simulations of the IWFM model application for a specified time interval. For instance, if the model timestep is a month and the model is simulated for a year, this procedure will perform simulations for 12 timesteps. Simulation time interval must be greater than or equal to the model simulation timestep. Simulation for an interval includes advancing the simulation time by one simulation timestep, reading in all the time-series input data for the simulated timestep, simulating the hydrology for one time step, printing out the simulation results, and advancing the system state in time. These steps are performed for each timestep within the simulation interval.

```
SUBROUTINE IW_Model_SimulateForAnInterval(iLen,cInterval,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLen
   CHARACTER(C_CHAR),INTENT(IN) :: cInterval(iLen)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_SimulateForAnInterval
```

`iLen` : Character length of the simulation time interval, `cInterval`

`cInterval` : Time interval to perform hydrologic simulations; acceptable intervals are

i. "1MIN"

ii. "2MIN"

iii. "3MIN"

iv. "4MIN"

v. "5MIN"

vi. "10MIN"

vii. “15MIN”

viii. “20MIN”

ix. “30MIN”

x. “1HOUR”

xi. “2HOUR”

xii. “3HOUR”

xiii. “4HOUR”

xiv. “6HOUR”

xv. “8HOUR”

xvi. “12HOUR”

xvii. “1DAY”

xviii. “1WEEK”

xix. “1MON”

xx. “1YEAR”

iStat : Error code; returns 0 if the procedure call was successful

8.1.120. IW_Model_SimulateAll

This procedure performs the hydrologic simulation for the entire simulation period of the IWFM model application.

```
SUBROUTINE IW_Model_SimulateAll(iStat)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_SimulateAll
```

iStat : Error code; returns 0 if the procedure call was successful

8.1.121. IW_Model_AdvanceTime

This procedure advances simulation time by one simulation timestep. For instance, if the simulation timestep is a day, and the current simulation time is 3/1/2000 (March 1st, 2000), calling this procedure advances simulation time to 3/2/2000. IWFM model simulations are time-aware; simulation times are used to read in the correct time-series input data as well as to print out the simulation results with the appropriate time stamp.

```
SUBROUTINE IW_Model_AdvanceTime(iStat)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_AdvanceTime
```

iStat : Error code; returns 0 if the procedure call was successful

8.1.122. IW_Model_ReadTSData

This procedure reads in all the time-series input data for the simulation time.

```
SUBROUTINE IW_Model_ReadTSData(iStat)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_ReadTSData
```

iStat : Error code; returns 0 if the procedure call was successful

8.1.123. IW_Model_ReadTSData_Overwrite

This procedure reads in all the time series input data for the simulation time but allows overwriting of some of the data with the user-specified information. This procedure is useful when some of the time-series information is simulated outside the IWFM model application. Currently, land-use acreages, surface water diversions and stream boundary inflows can be overwritten. For instance, land-use acreages can be calculated using an agricultural economics model, or surface water diversions and/or stream boundary inflows can be calculated using a surface water allocation model. Once these data are computed outside the IWFM model application, they can be passed to the API to overwrite the values read from the time-series input data files. If a particular dataset is not going to be overwritten, the size of the input array argument for that data is specified as 0 (zero).

```
SUBROUTINE IW_Model_ReadTSData_Overwrite(iNLandUse,iNSubregions,     &
             rRegionLUAreas,iNDiversions,iDiversions,rDiversions, &
             iNStrmInflows,iStrmInflowIDs,rStrmInflows,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNLandUse,iNSubregions,iNDiversions,  &
                                iNStrmInflows
   INTEGER(C_INT),INTENT(IN) :: iDiversions(iNDiversions),  &
                                iStrmInflows(iNStrmInflows)
   REAL(C_DOUBLE),INTENT(IN) :: rRegionLUAreas(iNSubregions,iNLandUse),  &
                                rDiversions(iNDiversions),               &
                                rStrmInflows(iNStrmInflows)
```

```
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_ReadTSData_Overwrite
```

`iNLandUse` : Total number of land-use categories (including non-ponded and ponded crops, urban, native and riparian vegetation) simulated; enter 0 if land-use acreages are not going to be overwritten

`iNSubregions` : Number of subregions; this value can be obtained by calling procedure `IW_Model_GetNSubregions` (see section 8.1.13); enter 0 if land-use acreages are not going to be overwritten

`rRegionLUAreas` : Land-use acreages at subregional level to overwrite values read from input file; these acreages are distributed to elements by the API using the element level area percentages; first set is for non-ponded crops, second set is for ponded crops, then for urban areas, native vegetation and finally riparian vegetation.

`iNDiversions` : Number of surface water diversions to be overwritten; enter 0 if diversion data will not be overwritten

`iDiversions` : Diversion indices at which diversion rates will be overwritten

`rDiversions` : Diversion rates that will overwrite the values read from the input file at diversions listed in the `iDiversions` argument

`iNStrmInflows` : Number of stream boundary inflows to be overwritten; enter 0 if stream boundary inflow data will not be overwritten

`iStrmInflows` : Stream inflow indices for which boundary inflow will be overwritten

`rStreamInflows` : Stream boundary inflows that will overwrite the values read from the input file for the inflows listed in the `iStrmInflows` argument

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.124. IW_Model_PrintResults

This procedure prints out all the simulation results at the end of a simulation time.

```
SUBROUTINE IW_Model_PrintResults(iStat)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_PrintResults
```

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.125. IW_Model_AdvanceState

This procedure advances the state of the hydrologic system in time (e.g. groundwater heads at current timestep are switched to groundwater heads at previous timestep).

```
SUBROUTINE IW_Model_AdvanceState(iStat)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_AdvanceState
```

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.126. IW_Model_IsStrmUpstreamNode

This procedure checks if a specified stream node .is located upstream from another specified stream node within the stream network of the IWFM model.

```
SUBROUTINE IW_Model_IsStrmUpstreamNode(iStrmNode1,iStrmNode2, &
             iUpstrm,iStat)
   INTEGER(C_INT),INTENT(IN) :: iStrmNode1,iStrmNode2
   INTEGER(C_INT),INTENT(OUT) :: iUpstrm,iStat
END SUBROUTINE IW_Model_IsStrmUpstreamNode
```

`iStrmNode1` : Stream node index that is being checked if it is upstream of another stream node with the index `iStrmNode2`

`iStrmNode2` : Stream node index that is being checked if it is downstream of another stream node with the index `iStrmNode1`

`iUpstrm` : Flag that indicates if stream node `iStrmNode1` is upstream of stream node `iStrmNode2`; 1 = `iStrmNode1` is upstream of `iStrmNode2`; 0 = `iStrmNodeID1` is not upstream of `iStrmNode2` (note that if `iStrmNode1` and `iStrmNode2` are on different stream networks, then `iUpstrm` = 0)

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.127. IW_Model_IsEndOfSimulation

This procedure allows the client to check if the end of simulation period has been reached.

```
SUBROUTINE IW_Model_IsEndOfSimulation(iEndOfSimulation,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iEndOfSimulation,iStat
END SUBROUTINE IW_Model_IsEndOfSimulation
```

`iEndOfSimulation` : Flag to check if end of simulation period is reached; 1 = end of simulation is reached, 0 = end of simulation is not reached

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.128. IW_Model_IsModelInstantiated

This procedure allows the client to check if a Model object is instantiated.

```
SUBROUTINE IW_Model_IsModelInstantiated(iInstantiated,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iInstantiated,iStat
END SUBROUTINE IW_Model_IsModelInstantiated
```

`iInstantiated` : Flag to check if a Model object is instantiated; 1 = Model object is instantiated, 0 = Model object is not instantiated

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.129. IW_Model_TurnSupplyAdjustOnOff

This procedure turns the automatic supply adjustment of diversions and pumping to meet agricultural and/or urban water demands on or off.

```
SUBROUTINE IW_Model_TurnSupplyAdjustOnOff(iDivAdjustOn,iPumpAdjustOn,iStat)
   INTEGER(C_INT),INTENT(IN) :: iDivAdjustOn,iPumpAdjustOn
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_TurnSupplyAdjustOnOff
```

`iDivAdjustOn` : Flag to turn on or off the automatic adjustment of diversions to meet agricultural and/or urban water demands; 1 = turn diversion adjustment on, 0 = turn diversion adjustment off

`iPumpAdjustOn` : Flag to turn on or off the automatic adjustment of pumping to meet agricultural and/or urban water demands; 1 = turn pumping adjustment on, 0 = turn pumping adjustment off

`iStat` : Error code; returns 0 if the procedure call was successful

8.1.130. IW_Model_RestorePumpingToReadValues

This procedure restores the pumping rates to the values read from the Pumping Rate input file. This procedure is useful when it is necessary to re-simulate the hydrologic system (e.g. when IWFM is linked to a reservoir operations model in an iterative fashion) at a given timestep with pumping adjustment is on and the pumping values need to be restored to their original values.

```
SUBROUTINE IW_Model_RestorePumpingToReadValues(iStat)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Model_RestorePumpingToReadValues
```

`iStat` : Error code; returns 0 if the procedure call was successful

8.2. Budget Group

These procedures are used to open, read data from and close Budget files generated by IWFM model applications. Budget files store all information that is necessary to

read and process the budget data to be printed to text files, or to be imported into HEC-DSS or spreadsheet files.

8.2.1. IW_Budget_OpenFile

This procedure opens a budget file generated by an IWFM model application to retrieve data from.

```
SUBROUTINE IW_Budget_OpenFile(cFileName,iLen,iStat)
   INTEGER(C_INT),INTENT(IN):: iLen
   CHARACTER(C_CHAR),INTENT(IN) :: cFileName(iLen)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Budget_OpenFile
```

`cFileName` : Full path, including the filename, of the budget file

`iLen` : Character length of the budget file path

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.2. IW_Budget_CloseFile

This procedure closes a previously opened budget file.

```
SUBROUTINE IW_Budget_CloseFile(iStat)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_Budget_CloseFile
```

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.3. IW_Budget_GetNLocations

This procedure retrieves the number of locations for which budget data exists. For instance, if the budget file is for groundwater budget, number of locations will be the number of subregions in the IWFM model application area; if it is lake budget file then the number of locations is the number of lakes simulated; for stream reach budget number of locations is the number of stream reaches, etc.

```
SUBROUTINE IW_Budget_GetNLocations(NLocations,iStat)
```

```
    INTEGER(C_INT),INTENT(OUT) :: NLocations,iStat
END SUBROUTINE IW_Budget_GetNLocations
```

`NLocations` : Number of locations for which budget data is stored in the budget file

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.4. IW_Budget_GetLocationNames

This procedure retrieves the location names (e.g. subregion names, lake names, stream reach names, etc., depending on the type of budget file) from the budget file

```
SUBROUTINE IW_Budget_GetLocationNames(cLocNames,iLenLocNames,NLocations,  &
             iLocArray,iStat)
    INTEGER(C_INT),INTENT(IN) :: iLenLocNames,NLocations
    CHARACTER(C_CHAR),INTENT(OUT) :: cLocNames(iLenLocNames)
    INTEGER(C_INT),INTENT(OUT) :: iLocArray(NLocations),iStat
END SUBROUTINE IW_Budget_GetLocationNames
```

`cLocNames` : List of the location names concatenated into a scalar string argument

`iLenLocNames` : Character length of the `cLocNames` argument which must be set to a large enough value to store all location names; a value of 30 times the number of locations is appropriate

`NLocations` : Number of locations for which budget data is stored in the budget file; this value can be obtained by calling procedure `IW_Budget_GetNLocations` (see section 8.2.3)

`iLocArray` : Character position array so that client software can separate the `cLocNames` scalar string argument into individual location names

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.5. IW_Budget_GetNTimeSteps

This procedure retrieves the number of timesteps for which budget data is available.

```
SUBROUTINE IW_Budget_GetNTimeSteps(NTimeSteps,iStat)
   INTEGER(C_INT),INTENT(OUT) :: NTimeSteps,iStat
END SUBROUTINE IW_Budget_GetNTimeSteps
```

NTimeSteps : Number of timesteps for which budget data is available

iStat : Error code; returns 0 if the procedure call was successful

8.2.6. IW_Budget_GetTimeSpecs

This procedure retrieves all the timestamps, incremented by the budget data time interval, from the beginning of the simulation period to the end. Budget data time interval is also returned.

```
SUBROUTINE IW_Budget_GetTimeSpecs(cDataDatesAndTimes,iLenDates,cInterval,  &
             iLenInterval,NData,iLocArray,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLenDates,iLenInterval,NData
   CHARACTER(C_CHAR),INTENT(OUT) :: cDataDatesAndTimes(iLenDates),  &
                                    cInterval(iLenInterval)
   INTEGER(C_INT),INTENT(OUT) :: iLocArray(NData),iStat
END SUBROUTINE IW_Budget_GetTimeSpecs
```

cDataDatesAndTimes: All the timestamps, incremented by the budget data time interval, starting from the beginning date of the budget data to the ending date; all timestamps are concatenated into a scalar string argument

iLenDates : Character length of cDataDatesAndTimes argument which must be set to 16 times NData (see below) or greater

cInterval : Budget data time interval

iLenInterval : Character length of cInterval argument which must be set to 8 or greater

NData : Number of timestamps being retrieved; it can be obtained by calling procedure IW_Budget_GetNTimeSteps (see section 8.2.5)

`iLocArray` : Character position array so that client software can separate the `cDataDatesAndTimes` scalar string argument into individual timestamps

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.7. IW_Budget_GetNTitleLines

This procedure retrieves the number of title lines for the water budget of a location. The title lines include the type of the budget (e.g. groundwater, stream flow, lake, etc.), name of the location for which budget data is processed, the units in which budget data is displayed and, if applicable, area of the location.

```
SUBROUTINE IW_Budget_GetNTitleLines(NTitles,iStat)
   INTEGER(C_INT),INTENT(OUT) :: NTitles,iStat
END SUBROUTINE IW_Budget_GetNTitleLines
```

`NTitles` : Number of title lines

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.8. IW_Budget_GetTitleLength

This procedure retrieves the character length of the title lines.

```
SUBROUTINE IW_Budget_GetTitleLength(iLen,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLen,iStat
END SUBROUTINE IW_Budget_GetTitleLength
```

`iLen` : Character length of the title lines

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.9. IW_Budget_GetTitleLines

This procedure retrieves the title lines for the budget data for a location to be displayed in the files (text, spreadsheet, etc.) where the budget data is being imported into.

```
SUBROUTINE IW_Budget_GetTitleLines(NTitles,iLocation,FactArea,LengthUnit,  &
              AreaUnit,VolumeUnit,iLenUnit,cAltLocName,iLenAltLocName,     &
              cTitles,iLenTitles,iLocArray,iStat)
    INTEGER(C_INT),INTENT(IN) :: NTitles,iLocation,iLenUnit, iLenTitles,
                                 iLenAltLocName
    REAL(C_DOUBLE),INTENT(IN) :: FactArea
    CHARACTER(C_CHAR),INTENT(IN) :: LengthUnit(iLenUnit),  &
                                    AreaUnit(iLenUnit),    &
                                    VolumeUnit(iLenUnit)
    CHARACTER(C_CHAR),INTENT(IN) :: cAltLocName(iLenAltLocName)
    CHARACTER(C_CHAR),INTENT(OUT) :: cTitles(iLenTitles)
    INTEGER(C_INT),INTENT(OUT) :: iLocArray(NTitles),iStat
END SUBROUTINE IW_Budget_GetTitleLines
```

`NTitles` : Number of titles; this value can be obtained by calling procedure `IW_Budget_GetNTitleLines` (see section 8.2.7)

`iLocation` : Location identification number (e.g. subregion number, lake number, stream reach number, etc. depending on the type of budget data being processed) for which titles are being retrieved

`FactArea` : Conversion factor to convert area related data in the budget output titles from simulation unit to output unit

`LengthUnit` : Conversion factor to convert length related data in the budget output titles from simulation unit to output unit

`AreaUnit` : Name of the area unit (e.g. "acres") to appear in the budget output titles

`VolumeUnit` : Name of the volume unit (e.g. "acre-feet") to appear in the budget output titles

`iLenUnit` : Character length of the `LengthUnit`, `AreaUnit` and `VolumeUnit` arguments

`cAltLocName` : Alternative name for the location specified by the client software that will overwrite the location name stored in the Budget file; if `cAltLocName` is empty then the original location name is not overwritten

`iLenAltLocName` : Character length of the `cAltLocName` argument

`cTitles` : Title lines, concatenated into a scalar string argument, for the location for which budget data is being retrieved

`iLenTitles` : Total character length of title lines; a value that is equal to `NTitles` times the character length of the title lines (obtained by calling `IW_Budget_GetTitleLength` procedure) is appropriate

`iLocArray` : Character position array so that client software can separate the `cTitles` scalar string argument into individual title lines

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.10. IW_Budget_GetNColumns

This procedure retrieves the number of budget data columns for a specified location.

```
SUBROUTINE IW_Budget_GetNColumns(iLoc,NColumns,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLoc
   INTEGER(C_INT),INTENT(OUT) :: NColumns,iStat
END SUBROUTINE IW_Budget_GetNColumns
```

`iLoc` : Location identification number (e.g. subregion number, lake number, stream reach number, etc.) for which budget data is being retrieved

`NColumns` : Number of budget data columns

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.11. IW_Budget_GetColumnHeaders

This procedure retrieves the budget data column headers. These headers can be used to identify budget data columns in text or spreadsheet files.

```
SUBROUTINE IW_Budget_GetColumnHeaders(iLoc,cColumnHeaders,               &
             iLenColumnHeaders,NColumns,LengthUnit,AreaUnit,VolumeUnit,  &
             iLenUnit,iLocArray,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLoc,iLenColumnHeaders,NColumns,iLenUnit
```

```
    CHARACTER(C_CHAR),INTENT(IN) :: LengthUnit(iLenUnit),  &
                                    AreaUnit(iLenUnit),    &
                                    VolumeUnit(iLenUnit)
    CHARACTER(C_CHAR),INTENT(OUT) :: cColumnHeaders(iLenColumnHeaders)
    INTEGER,INTENT(OUT) :: iLocArray(NColumns),iStat
END SUBROUTINE IW_Budget_GetColumnHeaders
```

`iLoc` : Location identification number (e.g. subregion number, lake number, stream reach number, etc.) for which budget data is being retrieved

`cColumnHeaders` : Budget data column headers concatenated into a scalar string argument

`iLenColumnHeaders` : Character length of the `cColumnHeader` argument; a value that is equal to the number of columns (obtained by calling `IW_Budget_GetNColumns` procedure (see section 8.2.10)) times 30 is appropriate

`NColumns` : Number of budget data columns which can be obtained by calling procedure `IW_Budget_GetNColumns` (see section 8.2.10)

`LengthUnit` : Length unit that will appear in the budget data column headers

`AreaUnit` : Area unit that will appear in the budget data column headers

`VolumeUnit` : Volume unit that will appear in the budget data column headers

`iLenUnit` : Character length of the `LengthUnit`, `AreaUnit` and `VolumeUnit` arguments

`iLocArray` : Character position array so that client software can separate the `cColumnHeaders` scalar string argument into individual budget data column headers

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.12. IW_Budget_GetValues

This procedure retrieves the budget data for selected budget columns for a location for a specified time period with a specified time interval. This procedure is able to return budget data with a time interval that is equal to or larger than the interval of the data stored in the budget file; e.g. budget data can be retrieved with a monthly time interval when the data is stored with a daily time interval in the budget file. The first column of data returned is the date of the budget data in MS Excel style Julian date format (with January 1, 1900 as Julian day 1).

```
SUBROUTINE IW_Budget_GetValues(iLoc,nReadCols,iReadCols,cDateAndTimeBegin, &
              cDateAndTimeEnd,iLenDateAndTime,cOutputInterval,iLenInterval,&
              rFact_LT,rFact_AR,rFact_VL,nTimes_In,Values,nTimes_Out,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLoc,nReadCols,iReadCols(nReadCols),   &
                                iLenDateAndTime,iLenInterval,nTimes_In
   CHARACTER(C_CHAR),INTENT(IN) :: cDateAndTimeBegin(iLenDateAndTime)
   CHARACTER(C_CHAR),INTENT(IN) :: cDateAndTimeEnd(iLenDateAndTime)
   CHARACTER(C_CHAR),INTENT(IN) :: cOutputInterval(iLenInterval)
   REAL(C_DOUBLE),INTENT(IN) :: rFact_LT,rFact_AR,rFact_VL
   REAL(C_DOUBLE),INTENT(OUT) :: Values(nReadCols+1,nTimes_In)
   INTEGER(C_INT),INTENT(OUT) :: nTimes_Out,iStat
END SUBROUTINE IW_Budget_GetValues
```

`iLoc` : Location identification number (e.g. subregion number, lake number, stream reach number, etc.) for which budget data is being retrieved

`nReadCols` : Number of budget data columns being retrieved

`iReadCols` : List of budget data columns being retrieved

`cDateAndTimeBegin` : Beginning date and time of the data retrieval period in IWFM timestamp format

`cDateAndTimeEnd` : Ending date and time of the data retrieval period in IWFM timestamp format

`iLenDateAndTime` : Character length of `cDateAndTimeBegin` and `cDateAndTimeEnd` arguments which must be 16 or greater

`cOutputInterval` : Budget data retrieval time interval which must be equal to or greater than the time interval of the data stored in the budget

file which can be obtained by calling procedure `IW_Budget_GetTimeSpecs` (see section 8.2.6); allowable time intervals are

i. "1MIN"
ii. "2MIN"
iii. "3MIN"
iv. "4MIN"
v. "5MIN"
vi. "10MIN"
vii. "15MIN"
viii. "20MIN"
ix. "30MIN"
x. "1HOUR"
xi. "2HOUR"
xii. "3HOUR"
xiii. "4HOUR"
xiv. "6HOUR"
xv. "8HOUR"
xvi. "12HOUR"
xvii. "1DAY"
xviii. "1WEEK"
xix. "1MON"
xx. "1YEAR"

`iLenInterval` : Character length of `cOutputInterval` argument

`rFact_LT` : Conversion factor that will be used to convert the unit of length stored in the budget file to output unit of length

`rFact_AR` : Conversion factor that will be used to convert the unit of area stored in the budget file to output unit of area

`rFact_VL` : Conversion factor that will be used to convert the unit of volume stored in the budget file to output unit of volume

`nTimes_In` : Maximum number of timesteps between the beginning and ending dates of the budget data retrieval; this value can be

calculated by calling `IW_GetNIntervals` procedure from the *Miscellaneous* procedures group; e.g. for data retrieval for a period of one non-leap year of budget data stored in the budget file with daily time interval, this value will be 365

`Values` : Retrieved budget data values; first column of the retrieved data represents the dates of the budget data in MS Excel Julian date format (January 1, 1900 is Julian day 1)

`nTimes_Out` : Actual number of timesteps for the budget data that is retrieved; e.g. if budget data stored at a daily interval is being retrieved at a monthly interval for a one year period this argument will be 12

`iStat` : Error code; returns 0 if the procedure call was successful

8.2.13. IW_Budget_GetValues_ForAColumn

This procedure retrieves the budget data for a single budget data column for a location for a specified time period with a specified time interval. This procedure is able to return budget data with a time interval that is equal to or larger than the interval of the data stored in the budget file; e.g. budget data can be retrieved with a monthly time interval when the data is stored with a daily time interval in the budget file.

```
SUBROUTINE IW_Budget_GetValues_ForAColumn(iLoc,iCol,cOutputInterval,       &
              iLenInterval,cOutputBeginDateAndTime,cOutputEndDateAndTime,  &
              iLenDateAndTime,rFact_LT,rFact_AR,rFact_VL,iDim_In,iDim_Out, &
              Dates,Values,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLoc,iCol,iLenInterval,iLenDateAndTime,  &
                                iDim_In
   CHARACTER(C_CHAR),INTENT(IN) :: cDateAndTimeBegin(iLenDateAndTime)
   CHARACTER(C_CHAR),INTENT(IN) :: cDateAndTimeEnd(iLenDateAndTime)
   CHARACTER(C_CHAR),INTENT(IN) :: cOutputInterval(iLenInterval)
   REAL(C_DOUBLE),INTENT(IN) :: rFact_LT,rFact_AR,rFact_VL
   INTEGER(C_INT),INTENT(OUT) :: iDim_Out,iStat
   REAL(C_DOUBLE),INTENT(OUT) :: Dates(iDim_In),Values(iDim_In)
END SUBROUTINE IW_Budget_GetValues_ForAColumn
```

`iLoc` : Location identification number (e.g. subregion number, lake number, stream reach number, etc.) for which budget data is being retrieved

`iCol` : Budget data column number that is being retrieved

`cOutputInterval` : Budget data retrieval time interval which must be equal to or greater than the time interval of the data stored in the budget file which can be obtained by calling procedure `IW_Budget_GetTimeSpecs` (see section 8.2.6); allowable time intervals are

i. "1MIN"
ii. "2MIN"
iii. "3MIN"
iv. "4MIN"
v. "5MIN"
vi. "10MIN"
vii. "15MIN"
viii. "20MIN"
ix. "30MIN"
x. "1HOUR"
xi. "2HOUR"
xii. "3HOUR"
xiii. "4HOUR"
xiv. "6HOUR"
xv. "8HOUR"
xvi. "12HOUR"
xvii. "1DAY"
xviii. "1WEEK"
xix. "1MON"
xx. "1YEAR"

`iLenInterval` : Character length of `cOutputInterval` argument

`cOutputBeginDateAndTime` : Beginning date and time of the data retrieval period in IWFM timestamp format

`cOutputEndDateAndTime` : Ending date and time of the data retrieval period in IWFM timestamp format

`iLenDateAndTime` : Character length of `cOutputBeginDateAndTime` and `cOutputEndDateAndTime` arguments which must be 16 or greater

`rFact_LT` : Conversion factor that will be used to convert the budget data length unit stored in the budget file to output unit of length

`rFact_AR` : Conversion factor that will be used to convert the budget data area unit stored in the budget file to output unit of area

`rFact_VL` : Conversion factor that will be used to convert the budget data volume unit stored in the budget file to output unit of volume

`iDim_In` : Maximum number of timesteps between the beginning and ending dates of the budget data retrieval; this value can be calculated by calling `IW_GetNIntervals` procedure from the *Miscellaneous* procedures group; e.g. for data retrieval for a period of one non-leap year of budget data stored in the budget file with daily time interval, this value will be 365

`iDim_Out` : Actual number of timesteps for the budget data that is retrieved; e.g. if budget data stored at a daily interval is being retrieved at a monthly interval for a one-year period this argument will be 12

`Dates` : Dates of the retrieved budget data in MS Excel Julian date format (January 1, 1900 is Julian day 1)

`Values` : Retrieved budget data

`iStat` : Error code; returns 0 if the procedure call was successful

8.3. ZBudget Group

These procedures are used to open, read data from and close Z-Budget files generated by IWFM model applications. Z-Budget files store all information that is necessary to read and process the budget data to be printed to text files, or to be imported into HEC-DSS or spreadsheet files.

8.3.1. IW_ZBudget_OpenFile

This procedure opens a Z-Budget file generated by an IWFM model application to retrieve data from.

```
SUBROUTINE IW_ZBudget_OpenFile(cFileName,iLen,iStat)
    INTEGER(C_INT),INTENT(IN):: iLen
    CHARACTER(C_CHAR),INTENT(IN) :: cFileName(iLen)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_ZBudget_OpenFile
```

`cFileName` : Full path, including the filename, of the Z-Budget file

`iLen` : Character length of the Z-Budget file path

`iStat` : Error code; returns 0 if the procedure call was successful

8.3.2. IW_ZBudget_CloseFile

This procedure closes a previously opened Z-Budget file.

```
SUBROUTINE IW_ZBudget_CloseFile(iStat)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_ZBudget_CloseFile
```

`iStat` : Error code; returns 0 if the procedure call was successful

8.3.3. IW_ZBudget_GenerateZoneList_FromFile

This procedure generates a list of zones and their neighboring zones based on data provided in a text file. This file must be in the same format as the *Zone Definition File* used by the Z-Budget post-processor.

```
SUBROUTINE IW_ZBudget_GenerateZoneList_FromFile(cFileName,iLen,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLen
   CHARACTER(C_CHAR),INTENT(IN) :: cFileName(iLen)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_ZBudget_GenerateZoneList_FromFile
```

`cFileName` : Full path, including the filename, of the Zone Definition File

`iLen` : Character length of the Zone Definition File path

`iStat` : Error code; returns 0 if the procedure call was successful

8.3.4. IW_ZBudget_GenerateZoneList

This procedure generates a list of zones and their neighboring zones based on data provided directly by the client software.

```
SUBROUTINE IW_ZBudget_GenerateZoneList(iZExtent,iNElems,iElems,iLayers,  &
             iZones,nZonesWithNames,iZonesWithNames,iLenZoneNames,      &
             cZoneNames,iLocArray,iStat)
   INTEGER(C_INT),INTENT(IN) :: iZExtent,iNElems,nZonesWithNames,  &
                                iElems(iNElems),iLayers(iNElems),  &
                                iZones(iNElems),                       &
                                iZonesWithNames(nZonesWithNames),      &
                                iLenZoneNames,iLocArray(nZonesWithNames)
   CHARACTER(C_CHAR),INTENT(IN) :: cZoneNames(iLenZoneNames)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_ZBudget_GenerateZoneList
```

`iZExtent` : Extent of the zone numbering defined by the zone extent codes that can be retrieved by the `IW_GetZoneExtentID_***` procedures from the *Miscellaneous* procedures group; zone numbering can either be defined for horizontal plane (`iZExtent` code retrieved by calling `IW_GetZoneExtentID_Horizontal` procedure) and will be used for all layers, or different zone numbering is specified for each

layer (iZExtent code retrieved by calling IW_GetZoneExtentID_Vertical procedure)

iNElems : Number of grid cells for which a zone number is assigned

iElems : List of grid cells for which a zone number is assigned

iLayers : List of layers (e.g. simulated aquifer layers) where grid cells listed in the iElems array belong to; if iZExtent is set to code for horizontal zone definition (see above) this information is not used and iLayers array can have any value

iZones : Zone numbers corresponding to grid cells listed in iElems array which belong to layers listed in iLayers array; if iZExtent is set to code for horizontal zone definition, zone numbers listed in iZones array are applied to all layers (grid cells that are not listed in iElems array are automatically assigned the default zone number of –99)

nZonesWithNames : Number of zones for which a name is specified; this number can be less than the actual number of zones defined

iLenZoneNames : Character length of all zone names concatenated into a single string scalar (see cZoneNames argument below); e.g. if there are two zones with names "MyZone" (6 characters) and "YourZone" (8 characters), this argument should have a value of 14

cZoneNames : String scalar that holds concatenated zone names; e.g. if there are two zones with names "MyZone" and "YourZone", this argument will have a value of "MyZoneYourZone"

iLocArray : Array that lists the character locations within the cZoneNames argument where the name of a zone starts; e.g. if there are two zones with names "MyZone" and "YourZone", this array will have its component values as iLocArray(1) = 1 and iLocArray(2) = 7

iStat : Error code; returns 0 if the procedure call was successful

8.3.5. IW_ZBudget_GetNZones

This procedure retrieves the number of zones.

```
SUBROUTINE IW_ZBudget_GetNZones(iNZones,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iNZones,iStat
END SUBROUTINE IW_ZBudget_GetNZones
```

`iNZones` : Number of zones

`iStat` : Error code; returns 0 if the procedure call was successful

8.3.6. IW_ZBudget_GetZoneList

This procedure retrieves the list of zone numbers.

```
SUBROUTINE IW_ZBudget_GetZoneList(iNZones,iZoneList,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNZones
   INTEGER(C_INT),INTENT(OUT) :: iZoneList(iNZones),iStat
END SUBROUTINE IW_ZBudget_GetZoneList
```

`iNZones` : Number of defined zones

`iZoneList` : List of zone numbers

`iStat` : Error code; returns 0 if the procedure call was successful

8.3.7. IW_ZBudget_GetNTimeSteps

This procedure retrieves the number of timesteps for which Z-Budget data is available.

```
SUBROUTINE IW_ZBudget_GetNTimeSteps(NTimeSteps,iStat)
   INTEGER(C_INT),INTENT(OUT) :: NTimeSteps,iStat
END SUBROUTINE IW_ZBudget_GetNTimeSteps
```

`NTimeSteps` : Number of timesteps for which Z-Budget data is available

`iStat` : Error code; returns 0 if the procedure call was successful

8.3.8. IW_ZBudget_GetTimeSpecs

This procedure retrieves all the timestamps, incremented by the Z-Budget data time interval, from the beginning of the simulation period to the end. Z-Budget data time interval is also returned.

```
SUBROUTINE IW_ZBudget_GetTimeSpecs(cDataDatesAndTimes,iLenDates,cInterval, &
             iLenInterval,NData,iLocArray,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLenDates,iLenInterval,NData
   CHARACTER(C_CHAR),INTENT(OUT) :: cDataDatesAndTimes(iLenDates),  &
                                    cInterval(iLenInterval)
   INTEGER(C_INT),INTENT(OUT) :: iLocArray(NData),iStat
END SUBROUTINE IW_ZBudget_GetTimeSpecs
```

`cDataDatesAndTimes`: All the timestamps, incremented by the Z-Budget data time interval, starting from the beginning date of the Z-Budget data to the ending date; all timestamps are concatenated into a scalar string argument

`iLenDates` : Character length of `cDataDatesAndTimes` argument which must be set to 16 times `NData` (see below) or greater

`cInterval` : Z-Budget data time interval

`iLenInterval` : Character length of `cInterval` argument which must be set to 8 or greater

`NData` : Number of timestamps being retrieved; it can be obtained by calling procedure `IW_ZBudget_GetNTimeSteps` (see section 8.3.7)

`iLocArray` : Character position array so that client software can separate the `cDataDatesAndTimes` scalar string argument into individual timestamps

`iStat` : Error code; returns 0 if the procedure call was successful

8.3.9. IW_ZBudget_GetColumnHeaders_General

This procedure retrieves the Z-Budget column headers (i.e. titles). For flow processes that simulate flow exchange between neighboring zones (e.g. groundwater process) the inflow and outflow columns are lumped into two columns (e.g. "*Inflows from Adjacent Zones*" and "*Outflows to Adjacent Zones*") instead of identifying inflows from and outflows to individual neighboring zones. These column headers apply to any zone regardless of the number of neighboring zones.

```
SUBROUTINE IW_ZBudget_GetColumnHeaders_General(NColumnsMax,AreaUnit,  &
              VolumeUnit,iLenUnit,iLenColumnHeaders,cColumnHeaders,  &
              NColumns,iLocArray,iStat)
   INTEGER(C_INT),INTENT(IN) :: NColumnsMax,iLenUnit,iLenColumnHeaders
   CHARACTER(C_CHAR),INTENT(IN) :: AreaUnit(iLenUnit),VolumeUnit(iLenUnit)
   CHARACTER(C_CHAR),INTENT(OUT) :: cColumnHeaders(iLenColumnHeaders)
   INTEGER(C_INT),INTENT(OUT) :: NColumns,iLocArray(NColumnsMax),iStat
END SUBROUTINE IW_ZBudget_GetColumnHeaders_General
```

`NColumnsMax` : Maximum number of columns that a Z-Budget file can have; set this argument large enough (e.g. 200) so that a large number of headers can be retrieved

`AreaUnit` : Area unit that will appear in the Z-Budget data column headers

`VolumeUnit` : Volume unit that will appear in the Z-Budget data column headers

`iLenUnit` : Character length of the `AreaUnit` and `VolumeUnit` arguments

`iLenColumnHeaders` : Character length of the `cColumnHeaders` argument; a value that is equal to `NColumnsMax` times 30 is appropriate

`cColumnHeaders` : Z-Budget data column headers concatenated into a scalar string argument

`NColumns` : Actual number of Z-Budget data column headers returned

`iLocArray` : Character position array so that client software can separate the `cColumnHeaders` scalar string argument into individual Z-Budget data column headers

`iStat` : Error code; returns 0 if the procedure call was successful

8.3.10. IW_ZBudget_GetColumnHeaders_ForAZone

This procedure retrieves the Z-Budget column headers (i.e. titles) for a specified zone for selected data columns. For flow processes that simulate flow exchange between neighboring zones (e.g. groundwater process), the column headers for inflows from and outflows to neighboring zones are listed separately for each neighboring zone. These columns are referred to as "*diversified columns*" since the inflows from and outflows to each neighboring zone are treated as separate columns as opposed to lumping them into two inflow and outflow columns as in `IW_ZBudget_GetColumnHeaders_General` procedure.

```
SUBROUTINE IW_ZBudget_GetColumnHeaders_ForAZone(iZone,NColumnsList,    &
              iColumnsList,NColumnsMax,AreaUnit,VolumeUnit,iLenUnit,   &
              iLenColumnHeaders,cColumnHeaders,NColumns,iLocArray,     &
              iColumnsListDiversified,iStat)
    INTEGER(C_INT),INTENT(IN) :: iZone,iLenColumnHeaders,NColumnsMax,  &
                                 iLenUnit,NColumnsList,                &
                                 iColumnsList(NColumnsList)
    CHARACTER(C_CHAR),INTENT(IN) :: AreaUnit(iLenUnit),VolumeUnit(iLenUnit)
    CHARACTER(C_CHAR),INTENT(OUT) :: cColumnHeaders(iLenColumnHeaders)
    INTEGER(C_INT),INTENT(OUT) :: NColumns,iLocArray(NColumnsMax),     &
                                  iColumnsListDiversified(NColumnsMax),iStat
END SUBROUTINE IW_ZBudget_GetColumnHeaders_ForAZone
```

`iZone` : Zone number for which diversified column headers are being retrieved

`NColumnsList` : Number of data columns for which column headers are being retrieved

`iColumnList` : List of data column indices for which column headers are being retrieved

`NColumnsMax` : Maximum number of headers that will be returned; set to a large value (e.g. 200) so that headers for a large number of columns can be retrieved

`AreaUnit` : Area unit that will appear in the Z-Budget data column headers

`VolumeUnit` : Volume unit that will appear in the Z-Budget data column headers

`iLenUnit` : Character length of the `AreaUnit` and `VolumeUnit` arguments

`iLenColumnHeaders` : Character length of the `cColumnHeaders` argument; a value that is equal to `NColumnsMax` times 30 is appropriate

`cColumnHeaders` : Z-Budget data column headers concatenated into a scalar string argument

`NColumns` : Actual number of Z-Budget data column headers returned

`iLocArray` : Character position array so that client software can separate the `cColumnHeaders` scalar string argument into individual Z-Budget data column headers

`iColumnsListDiversified` : Column indices for diversified columns; these indices can later be used to retrieve flow data for diversified columns

`iStat` : Error code; returns 0 if the procedure call was successful

8.3.11. IW_ZBudget_GetZoneNames

This procedure retrieves the zone names.

```
SUBROUTINE IW_ZBudget_GetZoneNames(iNZones,iLenZoneNames,cZoneNames,  &
             iLocArray,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNZones,iLenZoneNames
   CHARACTER(C_CHAR),INTENT(OUT) :: cZoneNames(iLenZoneNames)
   INTEGER(C_INT),INTENT(OUT) :: iLocArray(iNZones),iStat
END SUBROUTINE IW_ZBudget_GetZoneNames
```

`iNZones` : Number of defined zones

`iLenZoneNames` : Character length of the returned `cZoneNames` arguments; a value that is equal to `iNZones` times 30 is appropriate

cZoneNames	:	Zone names concatenated into a scalar string argument
iLocArray	:	Character position array so that client software can separate the cZoneNames scalar string argument into individual zone names
iStat	:	Error code; returns 0 if the procedure call was successful

8.3.12. IW_ZBudget_GetNTitleLines

This procedure retrieves the number of title lines for the Z-Budget output for a zone. The title lines include the IWFM version number that was used to generate the raw Z-Budget file, type of the Z-Budget (e.g. groundwater, root zone, land and water use, unsaturated zone), name of the zone for which the Z-Budget data is processed, the units in which the Z-Budget data is displayed and the area of the zone.

```
SUBROUTINE IW_ZBudget_GetNTitleLines(iNTitles,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iNTitles,iStat
END SUBROUTINE IW_ZBudget_GetNTitleLines
```

iNTitles	:	Number of title lines
iStat	:	Error code; returns 0 if the procedure call was successful

8.3.13. IW_ZBudget_GetTitleLines

This procedure retrieves the title lines for the Z-Budget data for a zone to be displayed in the files (text, spreadsheet, etc.) where the Z-Budget data is being imported into.

```
SUBROUTINE IW_ZBudget_GetTitleLines(iNTitles,iZone,rFact_AR,cUnit_AR,  &
              cUnit_VL,iLenUnit,cTitles,iLenTitles,iLocArray,iStat)
    INTEGER(C_INT),INTENT(IN) :: iNTitles,iZone,iLenUnit,iLenTitles
    REAL(C_DOUBLE),INTENT(IN) :: rFact_AR
    CHARACTER(C_CHAR),INTENT(IN) :: cUnit_AR(iLenUnit),cUnit_VL(iLenUnit)
    CHARACTER(C_CHAR),INTENT(OUT) :: cTitles(iLenTitles)
    INTEGER(C_INT),INTENT(OUT) :: iLocArray(iNTitles),iStat
END SUBROUTINE IW_Budget_GetTitleLines
```

iNTitles	:	Number of titles; this value can be obtained by calling procedure IW_ZBudget_GetNTitleLines (see section 8.3.12)
iZone	:	Zone number for which titles being retrieved
rFact_AR	:	Conversion factor to convert zone area from simulation unit of area to output unit
cUnit_AR	:	Name of the area unit (e.g. "acres") to appear in the Z-Budget output titles
cUnit_VL	:	Name of the volume unit (e.g. "acre-feet") to appear in the Z-Budget output titles
iLenUnit	:	Character length of the cUnit_AR and cUnit_VL arguments
cTitles	:	Title lines, concatenated into a scalar string argument, for the zone for which Z-Budget titles are being retrieved
iLenTitles	:	Total character length of title lines; a value that is equal to iNTitles times 200 is appropriate
iLocArray	:	Character position array so that client software can separate the cTitles scalar string argument into individual title lines
iStat	:	Error code; returns 0 if the procedure call was successful

8.3.14. IW_ZBudget_GetValues_ForSomeZones_ForAnInterval

This procedure is used to retrieve specified zone flow values for a specified list of zones for a given time interval.

```
SUBROUTINE IW_ZBudget_GetValues_ForSomeZones_ForAnInterval(iNZones,iZones, &
             iNDiversifiedReadColsMax,iDiversifiedReadCols,                &
             cDateAndTimeBegin,iLenDateAndTime,cOutputInterval,            &
             iLenInterval,rFact_AR,rFact_VL,rValues,iStat)
   INTEGER(C_INT),INTENT(IN) :: iNZones,iZones(iNZones),                   &
                  iNDiversifiedReadColsMax,                                 &
                  iDiversifiedReadCols(iNDiversifiedReadColsMax,iNZones), &
                  iLenDateAndTime,iLenInterval
   CHARACTER(C_CHAR),INTENT(IN) :: cDateAndTimeBegin(iLenDateAndTime),  &
                                   cOutputInterval(iLenInterval)
   REAL(C_DOUBLE),INTENT(IN) :: rFact_AR,rFact_VL
```

```
    REAL(C_DOUBLE),INTENT(OUT) :: rValues(iNDiversifiedReadColsMax,iNZones)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_ZBudget_GetValues_ForSomeZones_ForAnInterval
```

`iNZones` : Number of zones for which data are being retrieved

`iZones` : List of zone numbers for which data are being retrieved

`iNDiversifiedReadColsMax` : Maximum number of data columns that will be read including the Time column; for flow processes that simulate flow exchange between zones, total number of data columns changes from zone to zone because of the differing number of neighboring zones. For instance, for groundwater process, aside from standard inflows and outflows such as pumping, recharge, etc., each zone also has inflows from and outflows to neighboring zones. This means that depending on zone definition, each zone will have different number of data columns to be retrieved. This argument must specify the maximum of the number of data columns any of the zones have.

`iDiversifiedReadCols` : List of data columns that are being retrieved; column number 1 is the Time column

`cDateAndTimeBegin` : Beginning date and time of the data retrieval period in IWFM timestamp format

`iLenDateAndTime` : Character length of `cDateAndTimeBegin` argument which must be 16 or greater

`cOutputInterval` : Data retrieval time interval which must be equal to or greater than the time interval of the data stored in the Z-Budget file which can be obtained by calling procedure `IW_ZBudget_GetTimeSpecs` (see section 8.3.8); allowable time intervals are

i. "1MIN"

ii. "2MIN"

iii. "3MIN"

iv. “4MIN”

v. “5MIN”

vi. “10MIN”

vii. “15MIN”

viii. “20MIN”

ix. “30MIN”

x. “1HOUR”

xi. “2HOUR”

xii. “3HOUR”

xiii. “4HOUR”

xiv. “6HOUR”

xv. “8HOUR”

xvi. “12HOUR”

xvii. “1DAY”

xviii. “1WEEK”

xix. “1MON”

xx. “1YEAR”

`iLenInterval` : Character length of `cOutputInterval` argument

`rFact_AR` : Conversion factor that will be used to convert the unit of area stored in the Z-Budget file to output unit of area

`rFact_VL` : Conversion factor that will be used to convert the unit of volume stored in the Z-Budget file to output unit of volume

`rValues` : Retrieved data values; first column of the retrieved data represents the dates of the Z-Budget data in MS Excel Julian date format (January 1, 1900 is Julian day 1)

`iStat` : Error code; returns 0 if the procedure call was successful

8.3.15. IW_ZBudget_GetValues_ForAZone

This procedure is used to retrieve specified Z-Budget data columns for a specified zone for a time period.

```
SUBROUTINE IW_ZBudget_GetValues_ForAZone(iZone,iNDiversifiedReadCols,  &
             iDiversifiedReadCols,cDateAndTimeBegin,cDateAndTimeEnd,  &
             iLenDateAndTime,cOutputInterval,iLenInterval,rFact_AR,   &
             rFact_VL,iNTimes_In,rValues,iNTimes_Out,iStat)
   INTEGER(C_INT),INTENT(IN) :: iZone,iNDiversifiedReadCols,
                                iDiversifiedReadCols(iNDiversifiedReadCols), &
                                iLenDateAndTime,iLenInterval,iNTimes_In
   CHARACTER(C_CHAR),INTENT(IN) :: cDateAndTimeBegin(iLenDateAndTime),  &
                                   cDateAndTimeEnd(iLenDateAndTime),    &
                                   cOutputInterval(iLenInterval)
   REAL(C_DOUBLE),INTENT(IN) :: rFact_AR,rFact_VL
   REAL(C_DOUBLE),INTENT(OUT) :: rValues(iNDiversifiedReadCols,iNTimes_In)
   INTEGER(C_INT),INTENT(OUT) :: iNTimes_Out,iStat
END SUBROUTINE IW_ZBudget_GetValues_ForAZone
```

`iZone` : Zone number for which data is being retrieved

`iNDiversifiedReadCols` : Number of columns for which data is being retrieved

`iDiversifiedReadCols` : List of column numbers for which data is being retrieved

`cDateAndTimeBegin` : Beginning date and time of the data retrieval period in IWFM timestamp format

`cDateAndTimeEnd` : Ending date and time of the data retrieval period in IWFM timestamp format

`iLenDateAndTime` : Character length of `cDateAndTimeBegin` and `cDateAndTimeEnd` arguments which must be 16 or greater

`cOutputInterval` : Data retrieval time interval which must be equal to or greater than the time interval of the data stored in the budget file which can be obtained by calling procedure `IW_ZBudget_GetTimeSpecs` (see section 8.3.8); allowable time intervals are

i. "1MIN"

ii. "2MIN"

iii. "3MIN"

iv. “4MIN”

v. “5MIN”

vi. “10MIN”

vii. “15MIN”

viii. “20MIN”

ix. “30MIN”

x. “1HOUR”

xi. “2HOUR”

xii. “3HOUR”

xiii. “4HOUR”

xiv. “6HOUR”

xv. “8HOUR”

xvi. “12HOUR”

xvii. “1DAY”

xviii. “1WEEK”

xix. “1MON”

xx. “1YEAR”

`iLenInterval` : Character length of `cOutputInterval` argument

`rFact_AR` : Conversion factor that will be used to convert the unit of area stored in the budget file to output unit of area

`rFact_VL` : Conversion factor that will be used to convert the unit of volume stored in the budget file to output unit of volume

`iNTimes_In` : Maximum number of timesteps between the beginning and ending dates of the Z-Budget data retrieval; this value can be calculated by calling `IW_GetNIntervals` procedure from the *Miscellaneous* procedures group; e.g. for data retrieval for a period of one non-leap year of Z-Budget data stored in the budget file with daily time interval, this value will be 365

`rValues` : Retrieved data values; first column of the retrieved data represents the dates of the budget data in MS Excel Julian date format (January 1, 1900 is Julian day 1)

`iNTimes_Out` : Actual number of timesteps for the Z-Budget data that is retrieved; e.g. if Z-Budget data stored at a daily interval is being retrieved at a monthly interval for a one-year period this argument will be 12

`iStat` : Error code; returns 0 if the procedure call was successful

8.4. Miscellaneous Group

These procedures allow client software to obtain codes and flags used by the IWFM API for the identification of location types, data unit types and flow destination types. There are also several utility procedures in this group.

8.4.1. IW_GetDataUnitTypeID_Length

This procedure retrieves the code IWFM API uses to indicate a data type with units of length.

```
SUBROUTINE IW_GetDataUnitTypeID_Length(iDataUnitTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iDataUnitTypeID,iStat
END SUBROUTINE IW_GetDataUnitTypeID_Length
```

`iDataUnitTypeID` : Code that indicates a data type with units of length

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.2. IW_GetDataUnitTypeID_Area

This procedure retrieves the code IWFM API uses to indicate a data type with units of area.

```
SUBROUTINE IW_GetDataUnitTypeID_Area(iDataUnitTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iDataUnitTypeID,iStat
END SUBROUTINE IW_GetDataUnitTypeID_Area
```

`iDataUnitTypeID` : Code that indicates a data type with units of area

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.3. IW_GetDataUnitTypeID_Volume

This procedure retrieves the code IWFM API uses to indicate a data type with units of volume.

```
SUBROUTINE IW_GetDataUnitTypeID_Volume(iDataUnitTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iDataUnitTypeID,iStat
END SUBROUTINE IW_GetDataUnitTypeID_Volume
```

`iDataUnitTypeID` : Code that indicates a data type with units of volume

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.4. IW_GetDataUnitTypeIDs

This procedure retrieves the codes IWFM API uses to indicate data types with units of either length, area or volume.

```
SUBROUTINE IW_GetDataUnitTypeIDs(iDataUnitTypeID_Length,  &
             iDataUnitTypeID_Area,iDataUnitTypeID_Volume,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iDataUnitTypeID_Length,  &
                                 iDataUnitTypeID_Area,    &
                                 iDataUnitTypeID_Volume,  &
                                 iStat
END SUBROUTINE IW_GetDataUnitTypeIDs
```

`iDataUnitTypeID_Length` : Code that indicates a data type with units of length

`iDataUnitTypeID_Area` : Code that indicates a data type with units of area

`iDataUnitTypeID_Volume` : Code that indicates a data type with units of volume

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.5. IW_GetLandUseTypeID_GenAg

This procedure retrieves the code IWFM API uses to indicate agricultural lands (i.e. non-ponded, rice or refuge lands).

```
SUBROUTINE IW_GetLandUseTypeID_GenAg(iLUTypeID_GenAg,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLUTypeID_GenAg,iStat &
END SUBROUTINE IW_GetLandUseTypeID_GenAg
```

iLUTypeID_GenAg	:	Code that is used for agricultural lands (i.e. non-ponded, rice or refuge lands)
iStat	:	Error code; returns 0 if the procedure call was successful

8.4.6. IW_GetLandUseTypeID_Urban

This procedure retrieves the code IWFM API uses to indicate urban lands.

```
SUBROUTINE IW_GetLandUseTypeID_Urban(iLUTypeID_Urb,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLUTypeID_Urb,iStat &
END SUBROUTINE IW_GetLandUseTypeID_Urban
```

iLUTypeID_Urb	:	Code that is used for urban lands
iStat	:	Error code; returns 0 if the procedure call was successful

8.4.7. IW_GetLandUseTypeID_NonPondedAg

This procedure retrieves the code IWFM API uses to indicate agricultural lands with non-ponded crops only.

```
SUBROUTINE IW_GetLandUseTypeID_NonPondedAg(iLUTypeID_NonPondedAg,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLUTypeID_NonPondedAg,iStat &
END SUBROUTINE IW_GetLandUseTypeID_NonPondedAg
```

iLUTypeID_NonPondedAg	:	Code that is used for agricultural lands with non-ponded crops only
iStat	:	Error code; returns 0 if the procedure call was successful

8.4.8. IW_GetLandUseTypeID_Rice

This procedure retrieves the code IWFM API uses to indicate agricultural lands with rice only.

```
SUBROUTINE IW_GetLandUseTypeID_Rice(iLUTypeID_Rice,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLUTypeID_Rice,iStat &
END SUBROUTINE IW_GetLandUseTypeID_Rice
```

iLUTypeID_Rice : Code that is used for agricultural lands with rice only

iStat : Error code; returns 0 if the procedure call was successful

8.4.9. IW_GetLandUseTypeID_Refuge

This procedure retrieves the code IWFM API uses to indicate refuge lands.

```
SUBROUTINE IW_GetLandUseTypeID_Refuge(iLUTypeID_Refuge,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLUTypeID_Refuge,iStat &
END SUBROUTINE IW_GetLandUseTypeID_Refuge
```

iLUTypeID_Refuge : Code that is used for refuge lands

iStat : Error code; returns 0 if the procedure call was successful

8.4.10. IW_GetLandUseTypeID_NVRV

This procedure retrieves the code IWFM API uses to indicate native and riparian vegetation lands.

```
SUBROUTINE IW_GetLandUseTypeID_NVRV(iLUTypeID_NVRV,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLUTypeID_NVRV,iStat &
END SUBROUTINE IW_GetLandUseTypeID_NVRV
```

iLUTypeID_NVRV : Code that is used for lands with native and riparian vegetation

iStat : Error code; returns 0 if the procedure call was successful

8.4.11. IW_GetLandUseTypeIDs

This procedure retrieves the codes IWFM API uses to indicate land use types.

```
SUBROUTINE IW_GetLandUseTypeIDs(iLUTypeID_GenAg,iLUTypeID_Urb,      &
            iLUTypeID_NonPondedAg,iLUTypeID_Rice,iLUTypeID_Refuge, &
```

```
            iLUTypeID_NVRV,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLUTypeID_GenAg,iLUTypeID_Urb,        &
                                 iLUTypeID_NonPondedAg,iLUTypeID_Rice, &
                                 iLUTypeID_Refuge,iLUTypeID_NVRV,iStat &
END SUBROUTINE IW_GetLandUseTypeIDs
```

`iLUTypeID_GenAg` : Code that is used for agricultural lands (i.e. non-ponded, rice or refuge lands)

`iLUTypeID_Urb` : Code that is used for urban lands

`iLUTypeID_NonPondedAg` : Code that is used for agricultural lands with non-ponded crops only

`iLUTypeID_Rice` : Code that is used for agricultural lands with rice only

`iLUTypeID_Refuge` : Code that is used for refuges

`iLUTypeID_NVRV` : Code that is used for lands with native and riparian vegetation

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.12. IW_GetLocationTypeID_Node

This procedure retrieves the code IWFM API uses to indicate a model feature which is a finite element grid node.

```
SUBROUTINE IW_GetLocationTypeID_Node(iLocationTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_Node
```

`iLocationTypeID` : Code that indicates a finite element grid node feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.13. IW_GetLocationTypeID_Element

This procedure retrieves the code IWFM API uses to indicate a model feature which is finite element grid cell.

```
SUBROUTINE IW_GetLocationTypeID_Element(iLocationTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_Element
```

`iLocationTypeID` : Code that indicates a finite element grid cell feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.14. IW_GetLocationTypeID_Subregion

This procedure retrieves the code IWFM API uses to indicate a model feature which is a subregion.

```
SUBROUTINE IW_GetLocationTypeID_Subregion(iLocationTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_Subregion
```

`iLocationTypeID` : Code that indicates a subregion feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.15. IW_GetLocationTypeID_Zone

This procedure retrieves the code IWFM API uses to indicate a model feature which is a zone (i.e. a user-specified collection of finite element grid cells).

```
SUBROUTINE IW_GetLocationTypeID_Zone(iLocationTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_Zone
```

`iLocationTypeID` : Code that indicates a zone feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.16. IW_GetLocationTypeID_StrmNode

This procedure retrieves the code IWFM API uses to indicate a model feature which is a stream node.

```
SUBROUTINE IW_GetLocationTypeID_StrmNode(iLocationTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_StrmNode
```

`iLocationTypeID` : Code that indicates a stream node feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.17. IW_GetLocationTypeID_StrmReach

This procedure retrieves the code IWFM API uses to indicate a model feature which is a stream reach.

```
SUBROUTINE IW_GetLocationTypeID_StrmReach(iLocationTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_StrmReach
```

`iLocationTypeID` : Code that indicates a stream reach feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.18. IW_GetLocationTypeID_Lake

This procedure retrieves the code IWFM API uses to indicate a model feature which is a lake.

```
SUBROUTINE IW_GetLocationTypeID_Lake(iLocationTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_Lake
```

`iLocationTypeID` : Code that indicates a lake feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.19. IW_GetLocationTypeID_SmallWatershed

This procedure retrieves the code IWFM API uses to indicate a model feature which is a small watershed.

```
SUBROUTINE IW_GetLocationTypeID_SmallWatershed(iLocationTypeID,iStat)
```

```
    INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_SmallWatershed
```

`iLocationTypeID` : Code that indicates a small watershed feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.20. IW_GetLocationTypeID_GWHeadObs

This procedure retrieves the code IWFM API uses to indicate a model feature which is a location where an IWFM model application prints out simulated groundwater heads.

```
SUBROUTINE IW_GetLocationTypeID_GWHeadObs(iLocationTypeID,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_GWHeadObs
```

`iLocationTypeID` : Code that indicates a groundwater head print-out location feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.21. IW_GetLocationTypeID_StrmHydObs

This procedure retrieves the code IWFM API uses to indicate a model feature which is a stream node location where an IWFM model application prints out the simulated stream flows.

```
SUBROUTINE IW_GetLocationTypeID_StrmHydObs(iLocationTypeID,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_StrmHydObs
```

`iLocationTypeID` : Code that indicates a stream flow print-out location feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.22. IW_GetLocationTypeID_SubsidenceObs

This procedure retrieves the code IWFM API uses to indicate a model feature which is location where an IWFM model application prints out the simulated subsidence.

```
SUBROUTINE IW_GetLocationTypeID_SubsidenceObs(iLocationTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_SubsidenceObs
```

`iLocationTypeID` : Code that indicates a subsidence print-out location feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.23. IW_GetLocationTypeID_TileDrainObs

This procedure retrieves the code IWFM API uses to indicate a model feature which is a tile drain with hydrograph print-out.

```
SUBROUTINE IW_GetLocationTypeID_TileDrainObs(iLocationTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID,iStat
END SUBROUTINE IW_GetLocationTypeID_TileDrainObs
```

`iLocationTypeID` : Code that indicates a tile drain feature with hydrograph print-out

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.24. IW_GetLocationTypeIDs

This procedure retrieves all the codes IWFM API uses to identify feature types for nodes, elements, subregions, zones, lakes, stream nodes and reaches and small watersheds as well as groundwater head, stream flow, subsidence and tile drain print-out locations.

```
SUBROUTINE IW_GetLocationTypeIDs(iLocationTypeID_Node,                  &
            iLocationTypeID_Element,iLocationTypeID_Subregion,          &
            iLocationTypeID_Zone,iLocationTypeID_Lake,                  &
            iLocationTypeID_StrmNode,iLocationTypeID_StrmReach,         &
            iLocationTypeID_TileDrainObs,iLocationTypeID_SmallWatershed,&
            iLocationTypeID_GWHeadObs,iLocationTypeID_StrmHydObs,       &
            iLocationTypeID_SubsidenceObs,iStat)
```

```
    INTEGER(C_INT),INTENT(OUT) :: iLocationTypeID_Node,            &
                                  iLocationTypeID_Element,         &
                                  iLocationTypeID_Subregion,       &
                                  iLocationTypeID_Zone,            &
                                  iLocationTypeID_Lake,            &
                                  iLocationTypeID_StrmNode,        &
                                  iLocationTypeID_StrmReach,       &
                                  iLocationTypeID_TileDrainObs,    &
                                  iLocationTypeID_SmallWatershed,  &
                                  iLocationTypeID_GWHeadObs,       &
                                  iLocationTypeID_StrmHydObs,      &
                                  iLocationTypeID_SubsidenceObs,   &
                                  iStat
END SUBROUTINE IW_GetLocationTypeIDs
```

`iLocationTypeID_Node` : Code that indicates a node feature

`iLocationTypeID_Element` : Code that indicates an element feature

`iLocationTypeID_Subregion` : Code that indicates a subregion feature

`iLocationTypeID_Zone` : Code that indicates a zone feature

`iLocationTypeID_Lake` : Code that indicates a lake feature

`iLocationTypeID_StrmNode` : Code that indicates a stream node feature

`iLocationTypeID_StrmReach` : Code that indicates a stream reach feature

`iLocationTypeID_TileDrainObs` : Code that indicates a tile drain feature with hydrograph print-out

`iLocationTypeID_SmallWatershed` : Code that indicates a small watershed feature

`iLocationTypeID_GWHeadObs` : Code that indicates a groundwater head print-out location feature

`iLocationTypeID_StrmHydObs` : Code that indicates a stream flow print-out location feature

`iLocationTypeID_SubsidenceObs` : Code that indicates a subsidence print-out location feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.25. IW_GetFlowDestTypeID_Outside

This procedure retrieves the code IWFM API uses to indicate outside of the model domain as the destination of flow from a hydrologic feature.

```
SUBROUTINE IW_GetFlowDestTypeID_Outside(iFlowDestTypeID,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iFlowDestTypeID,iStat
END SUBROUTINE IW_GetFlowDestTypeID_Outside
```

iFlowDestTypeID	:	Code that indicates outside the model domain as the destination of flow from a hydrologic feature
iStat	:	Error code; returns 0 if the procedure call was successful

8.4.26. IW_GetFlowDestTypeID_Element

This procedure retrieves the code IWFM API uses to indicate a finite element grid cell as the destination of flow from a hydrologic feature.

```
SUBROUTINE IW_GetFlowDestTypeID_Element(iFlowDestTypeID,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iFlowDestTypeID,iStat
END SUBROUTINE IW_GetFlowDestTypeID_Element
```

iFlowDestTypeID	:	Code that indicates a finite element grid cell as the destination of flow from a hydrologic feature
iStat	:	Error code; returns 0 if the procedure call was successful

8.4.27. IW_GetFlowDestTypeID_ElementSet

This procedure retrieves the code IWFM API uses to indicate a group of finite element grid cells as the destination of flow from a hydrologic feature.

```
SUBROUTINE IW_GetFlowDestTypeID_ElementSet(iFlowDestTypeID,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iFlowDestTypeID,iStat
END SUBROUTINE IW_GetFlowDestTypeID_ElementSet
```

`iFlowDestTypeID` : Code that indicates a group of finite element grid cells as the destination of flow from a hydrologic feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.28. IW_GetFlowDestTypeID_GWElement

This procedure retrieves the code IWFM API uses to indicate that flow from a hydrologic feature flows into the groundwater at a finite element grid cell.

```
SUBROUTINE IW_GetFlowDestTypeID_GWElement(iFlowDestTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iFlowDestTypeID,iStat
END SUBROUTINE IW_GetFlowDestTypeID_GWElement
```

`iFlowDestTypeID` : Code that indicates groundwater at a finite element grid cell as the destination of flow from a hydrologic feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.29. IW_GetFlowDestTypeID_StrmNode

This procedure retrieves the code IWFM API uses to indicate a stream node as the destination of flow from a hydrologic feature.

```
SUBROUTINE IW_GetFlowDestTypeID_StrmNode(iFlowDestTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iFlowDestTypeID,iStat
END SUBROUTINE IW_GetFlowDestTypeID_StrmNode
```

`iFlowDestTypeID` : Code that indicates a stream node as the destination of flow from a hydrologic feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.30. IW_GetFlowDestTypeID_Lake

This procedure retrieves the code IWFM API uses to indicate a lake as the destination of flow from a hydrologic feature.

```
SUBROUTINE IW_GetFlowDestTypeID_Lake(iFlowDestTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iFlowDestTypeID,iStat
END SUBROUTINE IW_GetFlowDestTypeID_Lake
```

iFlowDestTypeID : Code that indicates a lake as the destination of flow from a hydrologic feature

iStat : Error code; returns 0 if the procedure call was successful

8.4.31. IW_GetFlowDestTypeID_Subregion

This procedure retrieves the code IWFM API uses to indicate a subregion as the destination of flow from a hydrologic feature.

```
SUBROUTINE IW_GetFlowDestTypeID_Subregion(iFlowDestTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iFlowDestTypeID,iStat
END SUBROUTINE IW_GetFlowDestTypeID_Subregion
```

iFlowDestTypeID : Code that indicates a subregion as the destination of flow from a hydrologic feature

iStat : Error code; returns 0 if the procedure call was successful

8.4.32. IW_GetFlowDestTypeIDs

This procedure retrieves all the codes that are used to indicate different types of model features as the destination of flows from hydrologic features.

```
SUBROUTINE IW_GetFlowDestTypeIDs(iFlowDestTypeID_Outside,         &
              iFlowDestTypeID_StrmNode,                           &
              iFlowDestTypeID_Element,iFlowDestTypeID_Lake,       &
              iFlowDestTypeID_Subregion,iFlowDestTypeID_GWElement,&
              iFlowDestTypeID_ElementSet,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iFlowDestTypeID_StrmNode,     &
                                 iFlowDestTypeID_Element,      &
                                 iFlowDestTypeID_Lake,         &
                                 iFlowDestTypeID_Subregion,    &
                                 iFlowDestTypeID_GWElement,    &
                                 iFlowDestTypeID_ElementSet,   &
                                 iStat
END SUBROUTINE IW_GetFlowDestTypeIDs
```

`iFlowDestTypeID_Outside` : Code that indicates outside the model domain as the destination of flow from a hydrologic feature

`iFlowDestTypeID_StrmNode` : Code that indicates a stream node as the destination of flow from a hydrologic feature

`iFlowDestTypeID_Element` : Code that indicates a grid cell as the destination of flow from a hydrologic feature

`iFlowDestTypeID_Lake` : Code that indicates a lake as the destination of flow from a hydrologic feature

`iFlowDestTypeID_Subregion` : Code that indicates a subregion as the destination of flow from a hydrologic feature

`iFlowDestTypeID_GWElement` : Code that indicates groundwater at a grid cell as the destination of flow from a hydrologic feature

`iFlowDestTypeID_ElementSet` : Code that indicates a group of grid cells as the destination of flow from a hydrologic feature

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.33. IW_GetSupplyTypeID_Diversion

This procedure retrieves the code IWFM API uses to indicate that the source of a water supply is a diversion.

```
SUBROUTINE IW_GetSupplyTypeID_Diversion(iSupplyTypeID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iSupplyTypeID,iStat
END SUBROUTINE IW_GetSupplyTypeID_Diversion
```

`iSupplyTypeID` : Code that indicates that the source of a water supply is a diversion

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.34. IW_GetSupplyTypeID_Well

This procedure retrieves the code IWFM API uses to indicate that the source of a water supply is a well.

```
SUBROUTINE IW_GetSupplyTypeID_Well(iSupplyTypeID,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iSupplyTypeID,iStat
END SUBROUTINE IW_GetSupplyTypeID_Well
```

`iSupplyTypeID` : Code that indicates that the source of a water supply is a well

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.35. IW_GetSupplyTypeID_ElemPump

This procedure retrieves the code IWFM API uses to indicate that the source of a water supply is element pumping.

```
SUBROUTINE IW_GetSupplyTypeID_ElemPump(iSupplyTypeID,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iSupplyTypeID,iStat
END SUBROUTINE IW_GetSupplyTypeID_ElemPump
```

`iSupplyTypeID` : Code that indicates that the source of a water supply is element pumping

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.36. IW_GetZoneExtentID_Horizontal

This procedure retrieves the code IWFM API uses to indicate that zones are defined in horizontal direction and they apply to all modeling layers in the vertical. Zone definitions are used for zone budget outputs that are retrieved from Z-Budget files.

```
SUBROUTINE IW_GetZoneExtentID_Horizontal(iZoneExtentID,iStat)
    INTEGER(C_INT),INTENT(OUT) :: iZoneExtentID,iStat
END SUBROUTINE IW_GetZoneExtentID_Horizontal
```

`iZoneExtentID` : Code that is used to indicate that zones are defined in horizontal

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.37. IW_GetZoneExtentID_Vertical

This procedure retrieves the code IWFM API uses to indicate that zones are defined both in horizontal and vertical directions. Zone definitions are used for zone budget outputs that are retrieved from Z-Budget files.

```
SUBROUTINE IW_GetZoneExtentID_Vertical(iZoneExtentID,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iZoneExtentID,iStat
END SUBROUTINE IW_GetZoneExtentID_Vertical
```

`iZoneExtentID` : Code that is used to indicate that zones are defined in both horizontal and vertical directions

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.38. IW_GetZoneExtentIDs

This procedure retrieves all codes IWFM API uses to indicate how zones are defined. Zone definitions are used for zone budget outputs that are retrieved from Z-Budget files.

```
SUBROUTINE IW_GetZoneExtentIDs(iZoneExtentID_Horizontal,                &
            iZoneExtentID_Vertical,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iZoneExtentID_Horizontal               &
                                 iZoneExtentID_Vertical,iStat
END SUBROUTINE IW_GetZoneExtentIDs
```

`iZoneExtentID_Horizontal` : Code that is used to indicate that zones are defined only in the horizontal direction

`iZoneExtentID_Vertical` : Code that is used to indicate that zones are defined both in horizontal and vertical directions

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.39. IW_GetBudgetTypeIDs

This procedure retrieves the codes IWFM API uses to indicate Budget types.

```
SUBROUTINE IW_GetBudgetTypeIDs(iBudgetTypeID_GW,iBudgetTypeID_RootZone,     &
             iBudgetTypeID_LWU,iBudgetTypeID_NPCrop_RZ,                     &
             iBudgetTypeID_NPCrop_LWU,iBudgetTypeID_PCrop_RZ,               &
             iBudgetTypeID_PCrop_LWU,iBudgetTypeID_UnsatZone,               &
             iBudgetTypeID_StrmNode,iBudgetTypeID_StrmReach,                &
             iBudgetTypeID_DivDetail,iBudgetTypeID_SWShed,                  &
             iBudgetTypeID_Lake,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iBudgetTypeID_GW,                  &
                                 iBudgetTypeID_RootZone,            &
                                 iBudgetTypeID_LWU,                 &
                                 iBudgetTypeID_NPCrop_RZ,           &
                                 iBudgetTypeID_NPCrop_LWU,          &
                                 iBudgetTypeID_PCrop_RZ,            &
                                 iBudgetTypeID_PCrop_LWU,           &
                                 iBudgetTypeID_UnsatZone,           &
                                 iBudgetTypeID_StrmNode,            &
                                 iBudgetTypeID_StrmReach,           &
                                 iBudgetTypeID_DivDetail,           &
                                 iBudgetTypeID_SWShed,              &
                                 iBudgetTypeID_Lake,                &
                                 iStat
END SUBROUTINE IW_GetBudgetTypeIDs
```

`iBudgetTypeID_GW` : Code used for groundwater Budget output

`iBudgetTypeID_RootZone` : Code used for root zone Budget output

`iBudgetTypeID_LWU` : Code used for land and water use Budget output

`iBudgetTypeID_NPCrop_RZ` : Code used for non-ponded crop specific root zone Budget output

`iBudgetTypeID_NPCrop_LWU`: Code used for non-ponded crop specific land and water use Budget output

`iBudgetTypeID_PCrop_RZ` : Code used for ponded crop specific root zone Budget output

`iBudgetTypeID_PCrop_LWU` : Code used for ponded crop specific land and water use Budget output

`iBudgetTypeID_UnsatZone` : Code used for unsaturated zone Budget output

`iBudgetTypeID_StrmNode` : Code used for stream node Budget output

`iBudgetTypeID_StrmReach` : Code used for stream reach Budget output

`iBudgetTypeID_DivDetail` : Code used for diversion detail Budget output

`iBudgetTypeID_SWShed` : Code used for small watershed Budget output

`iBudgetTypeID_Lake` : Code used for lake Budget output

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.40. IW_GetZBudgetTypeIDs

This procedure retrieves the codes IWFM API uses to indicate Z-Budget types.

```
SUBROUTINE IW_GetZBudgetTypeIDs(iZBudgetTypeID_GW,iZBudgetTypeID_RootZone, &
             iZBudgetTypeID_LWU,iZBudgetTypeID_UnsatZone,iStat)
   INTEGER(C_INT),INTENT(OUT) :: iZBudgetTypeID_GW,            &
                                 iZBudgetTypeID_RootZone,      &
                                 iZBudgetTypeID_LWU,           &
                                 iZBudgetTypeID_UnsatZone,iStat
END SUBROUTINE IW_GetZBudgetTypeIDs
```

`iZBudgetTypeID_GW` : Code that is used for groundwater Z-Budget output

`iZBudgetTypeID_RootZone` : Code that is used for rot zone Z-Budget output

`iZBudgetTypeID_LWU` : Code that is used for land and water use Z-Budget output

`iZBudgetTypeID_UnsatZone` : Code that is used for unsaturated zone Z-Budget output

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.41. IW_GetVersion

This procedure retrieves the version number for the IWFM API.

```
SUBROUTINE IW_GetVersion(iLen,cVer,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLen
```

```
    CHARACTER(C_CHAR),INTENT(OUT) :: cVer(iLen)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_GetVersion
```

iLen : Character length of the version number; a large enough value such as 500 is appropriate

cVer : Version number of the IWFM API

iStat : Error code; returns 0 if the procedure call was successful

8.4.42. IW_GetNIntervals

This procedure calculates the number of time intervals with a specified length between two dates; e.g. number of days between January 1, 1990 and December 31, 2000.

```
SUBROUTINE IW_GetNIntervals(cBeginDateAndTime,cEndDateAndTime,
              iLenDateAndTime,cInterval,iLenInterval,NIntervals,iStat)
    INTEGER(C_INT),INTENT(IN) :: iLenDateAndTime,iLenInterval
    CHARACTER(C_CHAR),INTENT(IN) :: cBeginDateAndTime(iLenDateAndTime),  &
                                    cEndDateAndTime(iLenDateAndTime),    &
                                    cInterval(iLenInterval)
    INTEGER(C_INT),INTENT(OUT) :: NIntervals,iStat
END SUBROUTINE IW_GetNIntervals
```

cBeginDateAndTime : Beginning date and time of the period in IWFM timestamp format

cEndDateAndTime : Ending date and time of the period in IWFM timestamp format

iLenDateAndTime : Character length of cBeginDateAndTime and cEndDateAndTime arguments which must be 16 or greater

cInterval : Time interval; acceptable values are

i. "1MIN"

ii. "2MIN"

iii. "3MIN"

iv. "4MIN"

v. "5MIN"

vi. “10MIN”

vii. “15MIN”

viii. “20MIN”

ix. “30MIN”

x. “1HOUR”

xi. “2HOUR”

xii. “3HOUR”

xiii. “4HOUR”

xiv. “6HOUR”

xv. “8HOUR”

xvi. “12HOUR”

xvii. “1DAY”

xviii. “1WEEK”

xix. “1MON”

xx. “1YEAR”

`iLenInterval` : Character length of `cInterval` argument

`NIntervals` : Number of time intervals defined by `cInterval` between two dates denoted by `cBeginDateAndTime` and `cEndDateAndTime` arguments

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.43. IW_IncrementTime

This procedure increments a time stamp by a specified time interval. For example, incrementing *12/31/2000_24:00* by *1MON* will lead to *01/31/2001_24:00*.

```
SUBROUTINE IW_IncrementTime(iLenDateAndTime,cDateAndTime,iLenInterval,  &
              cInterval,iNCount,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLenDateAndTime,iLenInterval,iNCount
   CHARACTER(C_CHAR),INTENT(INOUT) :: cDateAndTime(iLenDateAndTime)
   CHARACTER(C_CHAR),INTENT(IN) :: cInterval(iLenInterval)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_IncrementTime
```

`iLenDateAndTime` : Character length of `cDateAndTime` argument

`cDateAndTime` : Date and time, given in IWFM timestamp format, that will be incremented by a time length of `cInterval`

`iLenInterval` : Character length of the `cInterval` argument

`cInterval` : Length of time interval by which `cDateAndTime` will be incremented; acceptable values are

i. "1MIN"
ii. "2MIN"
iii. "3MIN"
iv. "4MIN"
v. "5MIN"
vi. "10MIN"
vii. "15MIN"
viii. "20MIN"
ix. "30MIN"
x. "1HOUR"
xi. "2HOUR"
xii. "3HOUR"
xiii. "4HOUR"
xiv. "6HOUR"
xv. "8HOUR"
xvi. "12HOUR"
xvii. "1DAY"
xviii. "1WEEK"
xix. "1MON"
xx. "1YEAR"

`iNCount` : Number of times `cDateAndTime` will be incremented by a time length of `cInterval`; negative values decrement `cDateAndTime`. For instance, incrementing *12/31/2000_24:00* by *1MON* 3 times (i.e. `iNCount` =3) will lead to *03/31/2001_24:00*; decrementing *12/31/2000_24:00* by *1MON* (i.e. `iNCount` = −1) will lead to *11/30/2000_24:00*

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.44. IW_IsTimeGreaterThan

This procedure compares two timestamps and checks if the first timestamp is in the future of the second timestamp.

```
SUBROUTINE IW_IsTimeGreaterThan(iLenDateAndTime,cDateAndTime1,  &
             cDateAndTime2,isGreaterThan,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLenDateAndTime
   CHARACTER(C_CHAR),INTENT(INOUT) :: cDateAndTime1(iLenDateAndTime),  &
                                      cDateAndTime2(iLenDateAndTime)
   INTEGER(C_INT),INTENT(OUT) :: isGreaterThan,iStat
END SUBROUTINE IW_IsTimeGreaterThan
```

`iLenDateAndTime` : Character length of the timestamps being compared

`cDateAndTime1` : First timestamp being compared

`cDateAndTime2` : Second timestamp being compared

`isGreaterThan` : If the first timestamp is in the future of the second timestamp, this argument will be set to 1, otherwise it will be set to −1

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.45. IW_SetLogFile

This procedure opens a text log file for IWFM API to print out error and warning messages.

```
SUBROUTINE IW_SetLogFile(iLen,cFileName,iStat)
   INTEGER(C_INT),INTENT(IN) :: iLen
   CHARACTER(C_CHAR),INTENT(IN) :: cFileName(iLen)
   INTEGER(C_INT),INTENT(OUT) :: iStat
END FUNCTION IW_SetLogFile
```

`iLen` : Character length of the log filename

`cFileName` : Log filename

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.46. IW_CloseLogFile

This procedure closes the log file opened for IWFM API to print out error and warning messages.

```
SUBROUTINE IW_CloseLogFile(iStat)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_CloseLogFile
```

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.47. IW_GetLastMessage

This procedure is used to retrieve the error message in case a procedure call from IWFM API returns an error code (`iStat`) other than 0.

```
SUBROUTINE IW_GetLastMessage(iLen,cErrorMessage,iStat)
    INTEGER(C_INT),INTENT(IN) :: iLen
    CHARACTER(C_CHAR),INTENT(INOUT) :: cErrorMessage(iLen)
    INTEGER(C_INT),INTENT(OUT) :: iStat
END SUBROUTINE IW_GetLastMessage
```

`iLen` : Character length of the error message; a value of 500 is appropriate

`cErrorMessage` : Error message that is generated by the IWFM API procedure that was unsuccessfully called last

`iStat` : Error code; returns 0 if the procedure call was successful

8.4.48. IW_LogLastMessage

This procedure is used to print the last error message (generated when a procedure call from IWFM API returns an error code (`iStat`) other than 0) to the message log file. The message log file is the text file that is opened using the `IW_SetLogFile` procedure (see section 8.4.45). If this procedure is called before `IW_SetLogFile` procedure, the error/warning message is printed to a default file named *Message.log*.

```
SUBROUTINE IW_LogLastMessage(iStat)
    INTEGER(C_INT),INTENT(OUT) :: iStat
```

```
END SUBROUTINE IW_LogLastMessage
```

iStat : Error code; returns 0 if the procedure call was successful

8.4.49. fooScalar

This procedure can be used to test the calling mechanisms used by a client software in passing or retrieving scalar integer and real numbers to and from the IWFM API.

```
SUBROUTINE fooScalar(iArg,dArg)
    INTEGER(C_INT),INTENT(INOUT) :: iArg
    REAL(C_DOUBLE),INTENT(INOUT) :: dArg
END SUBROUTINE fooScalar
```

iArg : Integer argument; if calling of this procedure from a client software is successful, iArg will be modified by multiplying its original value by 2 (i.e. if passed iArg = 2, retrieved iArg = 4)

dArg : Real argument; if calling of this procedure from a client software is successful, dArg will be modified by multiplying its original value by 2 (i.e. if passed dArg = 2.0, retrieved dArg = 4.0)

8.4.50. foo1DArray

This procedure can be used to test the calling mechanisms used by a client software in passing or retrieving one-dimensional integer and real arrays to and from the IWFM API.

```
SUBROUTINE foo1DArray(iArrayDim,iArray,idArrayDim,dArray)
    INTEGER(C_INT),INTENT(IN) :: iArrayDim,idArrayDim
    INTEGER(C_INT),INTENT(INOUT) :: iArray(iArrayDim)
    REAL(C_DOUBLE),INTENT(INOUT) :: dArray(idArrayDim)
END SUBROUTINE foo1DArray
```

iArrayDim : Dimension of the integer array, iArray

iArray : Integer array; if calling of this procedure from a client software is successful, all components of the integer array will have a value of 5

dArrayDim : Dimension of the real array, dArray

dArray : Real array; if calling of this procedure from a client software is successful, all components of the real array will have a value of 3.2

8.4.51. foo2DArray

This procedure can be used to test the calling mechanisms used by a client software in passing or retrieving two-dimensional integer and real arrays to and from the IWFM API. When calling this procedure, care must be taken if the client software uses row-major ordering of multi-dimensional arrays (Fortran uses column-major ordering).

```
SUBROUTINE foo2DArray(iDim1,iDim2,iArray,idDim1,idDim2,dArray)
    INTEGER(C_INT),INTENT(IN) :: iDim1,iDim2,idDim1,idDim2
    INTEGER(C_INT),INTENT(INOUT) :: iArray(iDim1,iDim2)
    REAL(C_DOUBLE),INTENT(INOUT) :: dArray(idDim1,idDim2)
END SUBROUTINE foo2DArray
```

iDim1 : Number of rows of the integer array, iArray; i.e. the size of its first dimension

iDim2 : Number of columns of the integer array, iArray; i.e. the size of its second dimension

iArray : Integer array; if calling of this procedure from a client software is successful, all columns will have the associated row number (e.g. all columns in the first row will have the value 1, all columns in the second row will have the value 2, etc.)

idDim1 : Number of rows of the real array, dArray; i.e. the size of its first dimension

idDim2 : Number of columns of the real array, dArray; i.e. the size of its second dimension

dArray : Real array; if calling of this procedure from a client software is successful, all columns will have the associated row number

(e.g. all columns in the first row will have the value 1.0, all columns in the second row will have the value 2.0, etc.)

8.4.52. fooStrPassed

This procedure can be used to test the calling mechanisms used by a client software in passing a string variable to the IWFM API. The API does not modify the value of this variable.

```
SUBROUTINE fooStrPassed(iLen,cStrPassed)
   INTEGER(C_INT),INTENT(IN) :: iLen
   INTEGER(C_CHAR),INTENT(IN) :: cStrPassed(iLen)
END SUBROUTINE fooStrPassed
```

`iLen` : Character length of the passed string variable, `cStrPassed`

`cStrPassed` : String variable with a character length of `iLen` that is passed to the API; if calling of this procedure from a client software is successful, the API creates a new text file with the name *IW_API_Test.txt* and prints the value of `cStrPassed` to this file

8.4.53. fooStrReceived

This procedure can be used to test the calling mechanisms used by a client software in retrieving a string variable to the IWFM API.

```
SUBROUTINE fooStrReceived(iLen,cStrRecvd)
   INTEGER(C_INT),INTENT(IN) :: iLen
   INTEGER(C_CHAR),INTENT(OUT) :: cStrRecvd(iLen)
END SUBROUTINE fooStrReceived
```

`iLen` : Character length of the string variable, `cStrRecvd`; its value should be 21 or more

`cStrRecvd` : String variable with a character length of `iLen` that is returned to the client software; if calling of this procedure from a client software is successful, this variable will return with a value `'This is another test!'`