Learning ISIS_DLL By Examples

Version 1.0
July 2001

Heitor Barbieri
BIREME/PAHO/WHO

Contents

1. Introduction

What is intended with this paper is to reduce the programming learning curve of of the ISIS_DLL. No theoretical introduction is given here, so it is not supposed to be a replacement of the official manual of the ISIS_DLL.
The examples here are divided in three levels (basic, intermediate and advanced) according to its complexity and which we suppose to cover the most usually situations of use of the ISIS_DLL.
We advise to read of the explanations that follows the examples and if some daughts remains, a read of the chapter describing the subject being explained in the official manual will suffice to explain any daughts.
The examples are written in a C-like language, Visual Basic examples are given in the ISIS_DLL manual.

- All the examples require the inclusion of isis001.* (bas, pas or h) file and also the isisdll.h (C/C++) or isis32.* (bas, pas) file.

- The TOCHAR macro only gets the address of a structure and casts it to a pointer of char, other programming languages only have to pass the address of the respective struture as a parameter of the function.

- Every ISIS_DLL function returns a value, a negative number indicates an error code.

BASIC LEVEL

2.1 Reading a record from a database.

```
(1) long a;
(2) long h;
(3) long r;
(4) long mfn = 1;

(5) a = IsisAppNew( );
(6) h = IsisSpaNew(a);
(7) r = IsisSpaMf(h, “master_name”);
(8) r = IsisRecRead(h, 0, mfn);
(9) r = IsisAppDelete(a);
```

This first example only reads the first record of a database named “master_name”. It does not show its fields (see next example).
In the line (5) we get the application handle, its is necessary to create a space (6) where the master (7) and/or inverted file will be declared, so lines (5) and (6) appear in all ISIS_DLL application and generally they are the first ones. Note that lines (7) and (8) refer to space created in (6) by the use of the space handle as the first parameter.
After we have declared the master (7) we can now read the first record (8) and store it into the default shelf – number zero (second parameter).
The last ISIS_DLL function called is always the IsisAppDelete function (9) because it makes all ISIS_DLL garbage collection, no other ISIS_DLL function can be called after that.

2.2 Reading and displaying the fields of a record.

```
(1)  long a;
(2)  long h;
(3)  long r;
(4)  long mfn = 1;
(5)  char area[MAXMFRL];

(6)  a = IsisAppNew( );
(7)  h = IsisSpaNew(a);
(8)  r = IsisSpaMf(h, “master_name”);
(9)  r = IsisRecRead(h, 0, mfn);
(10) r = IsisRecDump(h, 0, area, MAXMFRL);
(11) printf(area);
(12) r = IsisAppDelete(a);
```

Here after reading the first record, line (9), we display the contents of its fields, line (10), into a pre allocated a memory area (5). Note that we specify the size of this area in the IsisRecDump function, in this case MAXMFRL, the constant defined in the isis001.* files. In the line (11) we print the fields of the first database record.

2.3 Displaying all records of a database.

```
(1) long a;
(2) long h;
(3) long r;
(4) long mfn;
(5) char area[MAXMFRL];
(6) struct IsisRecControl controlStructure;

(7) a = IsisAppNew( );
(8) h = IsisSpaNew(a);
(9) r = IsisSpaMf(h, “master_name”);

(10) r = IsisRecControlMap(h, TOCHAR(controlStructure))

(11) for (mfn = 1; mfn < controlStructure.nxtmfn; mfn++)  {
(12)          r = IsisRecRead(h, 0, mfn);
(13)          r = IsisRecDump(h, 0, area, MAXMFRL);
(14)          printf(area);
(15) }

(16) r = IsisAppDelete(a);
```

In this example we read the control record, line (10), to figure out the number of records in the data base. Note that the IsisRecControlMap function requires as a parameter the IsisRecControl structure declared in the line (6). In line 11 we loop all the records of the database showing its contents.

2.4 Formating all records of a database.

```
(1) long a;
(2) long h;
(3) long r;
(4) long mfn;
(5) char area[MAXMFRL];
```

```
(6) struct IsisRecControl controlStructure;

(7) a = IsisAppNew( );
(8) h = IsisSpaNew(a);
(9) r = IsisSpaMf(h, “master_name”);
(10) r = IsisSpaPft(h, “@format_specification_file”);

(11) r = IsisRecControlMap(h, TOCHAR(controlStructure))

(12) for (mfn = 1; mfn < controlStructure.nxtmfn; mfn++)  {
(13)          r = IsisRecRead(h, 0, mfn);
(14)          r = IsisRecFormat(h, 0, area, MAXMFRL);
(15)          printf(area);
(16) }

(17) r = IsisAppDelete(a);
```

This examples shows how to format the record instead of only dump its contents. In the line (10) we declare the format specification that will be used by the IsisRecFormat function (14) that is very similar to the IsisRecDump function. Note that the format specification (10) used here is in the file “format_specification_file” so it needs to be preceded by a `@´ symbol to tell the function that it is the name of a file and not the format specification itself. If we need to change the format specification we have to call the IsisSpaPft function again.

2.5 Updating fields of a record.

```
(1) long a;
(2) long h;
(3) long r;
(4) long mfn = 1;

(5) a = IsisAppNew( );
(6) h = IsisSpaNew(a);
(7) r = IsisSpaMf(h, “master_name”);
(8) r = IsisRecRead(h, 0, mfn);
(9) r = IsisRecFieldUpdate(h, 0, “d100  a100#new field#”);
(10) r = IsisRecWrite(h, 0);
(11) r = IsisAppDelete(a);
```

This example shows how to change the contents of a record, line (9), through the use of a language described in the ISIS_DLL manual (it is the same used by the mx software). Here we are deleting all fields with tag 100 and adding a new

occurrence of this field having the content of “new field” (a100 #new field#). Note that the change is done in memory only, to save the changes it is necessary to write the record in the database, line (10). Again, we are always using the default record shelf, shelf number zero.

2.6 Creating a new record and updating its fields.

```
(1)  long a;
(2)  long h;
(3)  long r;
(4)  long mfn;

(5)  a = IsisAppNew( );
(6)  h = IsisSpaNew(a);
(7)  r = IsisSpaMf(h, “master_name”);
(8)  mfn = IsisRecNew(h, 0);
(9)  r = IsisRecFieldUpdate(h, 0, “d100  a100#new field#”);
(10)  r = IsisRecWrite(h, 0);
(11)  r = IsisAppDelete(a);
```

In the previous example we updated the fields of an existing record, here we created a new record , line (8), before updating its fields. The new record is created in the database and not only in the memory, so it is not necessary to write it, we only did that, line (10), because we change its fields.

2.7 Creating a new master and new records.

```
(1)  long a;
(2)  long h;
(3)  long r;
(4)  long mfn;
(5)  long counter;

(6)  a = IsisAppNew( );
(7)  h = IsisSpaNew(a);
(8)  r = IsisSpaMf(h, “master_name”);

(9)  r = IsisSpaMfCreate(h);

(10)  for (counter = 0; counter < 10; counter++) {
(11)          mfn = IsisRecNew(h, 0);
(12)          r = IsisRecFieldUpdate(h, 0, “d100  a100#new field#
(13)                                          a200#other new field#”);
(13)          r = IsisRecWrite(h, 0);
(14)  }
```

```
(15) r = IsisAppDelete(a);
```

Here, we create a new database, line (9), after declaring it, line (8). In the line (10) we loop 10 times where we create 10 new records, line (11).

2.8 Searching a boolean expression.

```
(1) long a;
(2) long h;
(3) long r;
(4) struct IsisSrcHeader searchStructure;

(5) a = IsisAppNew( );
(6) h = IsisSpaNew(a);
(7) r = IsisSpaMf(h, “master_name”);
(8) r = IsisSpaIf(h, “inverted_name”);

(9) r = IsisSrcSearch(h, 0, “bol_expression”,
                                        TOCHAR(searchStructure));

(10) r = IsisAppDelete(a);
```

To search a boolean expression with the ISIS_DLL it is necessary to declare a search structure, IsisSrcHeader – line (4) - and then call the IsisSrcSearch function – line (9) – with the boolean expression, here “bol_expression”, and the search structure. The log of the search will be stored in the default log search – number zero (second parameter). Note here that it is necessary to declare the inverted file where the search will be executed, line (8).

2.9 Displaying the records retrieved from a search.

```
(1) long a;
(2) long h;
(3) long r;
(4) long counter;
(5) struct IsisSrcHeader searchStructure;
(6) struct IsisSrcMfn mfnStructure;
(7) char area[MAXMFRL];

(8) a = IsisAppNew( );
(9) h = IsisSpaNew(a);
```

```
(10) r = IsisSpaMf(h, “master_name”);
(11) r = IsisSpaIf(h, “inverted_name”);

(12) r = IsisSrcSearch(h, 0, “bol_expression”,
                              TOCHAR(searchStructure));

(13) for (counter = 0; counter < searchStructure.recs; counter++) {
(14)         r = IsisSrcMfnMap(a, 0, 0, 1, 1, TOCHAR(mfnStructure));
(15)         r = IsisRecRead(h, 0, mfnStructure.mfn);
(16)         r = IsisRecDump(h, 0, area, MAXMFRL);
(17)         printf(area);
(18) }

(19) r = IsisAppDelete(a);
```

After the search has been done, line (12), we want to shows its results. To do that we retrieve all the mfs that satisfy the seach via the IsisSrcMfnMap function, line (14). This function requires the application handle as its first parameter, than the log file number (we are using the default – number zero), the search number (zero means the last one), the range position of mfns (here the first mfn until the first mfn) and the mfn structure declared in the line (6) as its last parameter.

If the range of mfns is greater than one then the last parameter of IsisSrcMfnMap function should be a pre-allocated array of IsisSrcMfn structures instead of the structure itself.

INTERMEDIATE LEVEL

3.1 Updating the inverted file from a record.

```
(1) long a;
(2) long h;
(3) long r;
(4) long mfn = 1;

(5) a = IsisAppNew( );
(6) h = IsisSpaNew(a);
(7) r = IsisSpaMf(h, “master_name”);
(8) r = IsisSpaIf(h, “inverted_name”);
(9) r = IsisSpaFst(h, “@fst_file_name”);

(10) r = IsisRecRead(h, 0, mfn);
(11) r = IsisRecFieldUpdate(h, 0, “d100  a100#new field#”);
(12) r = IsisRecWrite(h, 0);
(12) r = IsisRecIfUpdate(h, mfn);

(13) r = IsisAppDelete(a);
```

To update a record in the inverted file, it is necessary to have its flag of modified record set. We change the record fields in the line (11) to set that flag. Before it we declared the field select table via IsisSpaFst function, line (9). Note the use of the ´@´ symbol to indicate that “fst_file_name” is the name of a file and not the fst specification itself. The record is updated in the line (12) with the IsisRecIfUpdate function which uses the mfn of the record and not its shelf number.

3.2 Creating and loading an inverted file.

```
(1) long a;
(2) long h;
(3) long r;

(4) a = IsisAppNew( );
(5) h = IsisSpaNew(a);
(6) r = IsisSpaMf(h, “master_name”);
(7) r = IsisSpaIf(h, “inverted_name”);
(8) r = IsisSpaFst(h, “@fst_file_name”);
```

```
(9) r = IsisIfCreate(h);

(10) r = IsisRecLnk(h, 1, MAXLONG);
(11) r = IsisLnkSort(h);
(12) r = IsisLnkIfLoad(h);

(13) r = IsisAppDelete(a);
```

This example shows how to create an inverted file, line (9), after it has been declared, line (7) and how to load the inverted file, lines (10) to (12).
In the line (10) we generate all the keys from mfn 1 to the end of the database (we used here a big number instead of checking the last mfn).
In the line (11) we sort all keys and then we load them using the IsisLnkIfLoad function, line (12).

3.3 Loading an inverted file in a customized way.

```
(1) long a;
(2) long h;
(3) long r;

(4) a = IsisAppNew( );
(5) h = IsisSpaNew(a);
(6) r = IsisSpaMf(h, “master_name”);
(7) r = IsisSpaIf(h, “inverted_name”);
(8) r = IsisSpaFst(h, “@fst_file_name”);

(9) r = IsisSpaStw(h, “@stw_file_name”);
(10) r = IsisSpaUcTab(h, “@uctab_file_name”);
(11) r = IsisSpaAcTab(h, “@actab_file_name”);

(12) r = IsisRecLnk(h, 1, MAXLONG);
(13) r = IsisLnkSort(h);
(14) r = IsisLnkIfLoad(h);

(15) r = IsisAppDelete(a);
```

This example shows how to customize your inverted file by specifying the stopwords (9) upper case character (10) and alphabetic character (11) tables. Note the use of the ´@` sign to distinguish the file name from the description itself.

3.4 Importing a database via an ISO-2709 file.

```
(1) long a;
(2) long h;
```

```
(3)  long r;

(4)  a = IsisAppNew( );
(5)  h = IsisSpaNew(a);
(6)  r = IsisSpaMf(h, “master_name”);
(7)  r = IsisSpaIsoIn(h, “iso_name”);

(8)  r = IsisSpaMfCreate(h);

(9)  while (true) {
(10)          r = IsisRecIsoRead(h, 0);
(11)          if (r > 0) {
(12)                  r = IsisRecWrite(h, 0);
(13)          } else {
(14)                  break;
(15)          }
(16) }

(17) r = IsisAppDelete(a);
```

The records imported from an ISO-2709 file can be added to an empty database (importing a database) or to a not empty database (appending records to a database). In this example, we choose the first option to create and/or initialize the “master_name” database in the line (8), after that, for each record in the ISO file, we import it – line (10) – and if the end of file hasn´t be reached – line (11) – the record is writed into the database – line (12).

3.5 Exporting a database via an ISO-2709 file.

```
(1)  long a;
(2)  long h;
(3)  long r;
(4)  struct IsisRecControl controlStructure;

(5)  a = IsisAppNew( );
(6)  h = IsisSpaNew(a);
(7)  r = IsisSpaMf(h, “master_name”);

(8)  r = IsisSpaIsoOut(h, “iso_name”);
(9)  r = IsisSpaIsoOutCreate(h);

(10) r = IsisRecControlMap(h, TOCHAR(controlStructure))

(11) for (mfn = 1; mfn < controlStructure.nxtmfn; mfn++)  {
(12)          r = IsisRecRead(h, 0, mfn);
(13)                  if (r == ZERO) {
```

```
(14)                    r = IsisRecIsoWrite(h, 0);
(15)                }
(16) }

(17) r = IsisAppDelete(a);
```

This example shows how to export a database to an ISO-2709 file. You can export the records to an empty file or appending them to an existing one. Here a ISO-2709 file is declared in the line (8) and it its createad and/or initialized in the line (9).
We figure out the number of records in the database by reading the control record in the line (10), then for each record in the database, we read it – line (12) – and export it – line (14).

3.6 Displaying the keys of an inverted file.

```
(1) long a;
(2) long h;
(3) long r;
(4) struct IsisTrmRead trmStructure;

(5) a = IsisAppNew( );
(6) h = IsisSpaNew(a);
(7) r = IsisSpaIf(h, “inverted_name”);

(8) trmStructure.key[0] = 0;
(9) r = IsisTrmShelfSize(h, 0, 0);

(10) while (true) {
(11)        r = IsisTrmReadNext(h, 0, TOCHAR(trmStructure));
(12)        if (r == ZERO) {
(13)            printf(trmStructure.key);
(14)        } else {
(15)            break;
(16)        }
(17) }

(18) r = IsisAppDelete(a);
```

To display the keys of an inverted file it is necessary to declare a IsisTrmRead structure, line (4), and initialize it with a key which next key will be the fist key of the database, we choose here to put an empty key (char 0) – line (8).
To increase the speed of loading the keys, we inform the ISIS_DLL not to load the entire postings but only the keys of the inverted file, this is done by setting the term shelf to size zero – line (9).

Then we loop lines (10) – (17), to read the next key – line (11) – and to print it – line (13).

3.7 Displaying the keys of an inverted file in a reverse order.

```
(1)  long a;
(2)  long h;
(3)  long r;
(4)  struct IsisTrmRead trmStructure;

(5)  a = IsisAppNew( );
(6)  h = IsisSpaNew(a);
(7)  r = IsisSpaIf(h, “inverted_name”);

(8)  trmStructure.key[0] = 255;
(9)  r = IsisTrmShelfSize(h, 0, 0);

(10) while (true) {
(11)        r = IsisTrmReadPrevious(h, 0, TOCHAR(trmStructure));
(12)        if (r == ZERO) {
(13)              printf(trmStructure.key);
(14)        } else {
(15)              break;
(16)        }
(17) }

(18) r = IsisAppDelete(a);
```

To display the keys of an inverted file it is necessary to declare a IsisTrmRead structure, line (4), and initialize it with a key which next key will be the last key of the database, we choose here to put the last ASCII key (char 255) – line (8).
To increase the speed of loading the keys, we inform the ISIS_DLL not to load the entire postings but only the keys of the inverted file, this is done by setting the term shelf to size zero – line (9).
then we loop lines (10) – (17), to read the next key – line (11) – and to print it – line (13).

ADVANCED LEVEL

4.1 Creating a database by using records of another one.

```
(1)  long a;
(2)  long h_from;
(3)  long h_to;
(4)  long r;
(5)  long mfn;
(6)  struct IsisRecControl controlStructure;

(7)  a = IsisAppNew( );

(8)  h_from = IsisSpaNew(a);
(9)  r = IsisSpaMf(h_from, “master_name_from”);

(10) h_to = IsisSpaNew(a);
(11) r = IsisSpaMf(h_to, “master_name_to”);

(12) r = IsisSpaMfCreate(h_to);

(13) r = IsisRecControlMap(h_from, TOCHAR(controlStructure))

(14) for (mfn = 1; mfn < controlStructure.nxtmfn; mfn++)  {
(15)          r = IsisRecRead(h_from, 0, mfn);
(16)          if (r == ZERO) {
(17)                   r = IsisRecNew(h_to, 0);
(18)                   r = IsisRecCopy(h_from, 0, h_to, 0);
(19)                   r = IsisRecWrite(h_to, 0);
(20)          }
(21) }

(22) r = IsisAppDelete(a);
```

In this example two IsisSpaces are created – lines (8) and (10) to manage two databases at the same time, so we have two space handles – h_from and h_to. We create the destination database in the line (12) and figure out the number of records of the source database (controlStructure.nxtmfn – 1) in the line (13). For each record of the source database (14) we read it (15), create a new record at the destination data base (17), copy the records between the databases (18) and write the new record (19). Note that the IsisRecCopy function uses the source and destination handles.

The same result could be obtained if we export the source database to an ISO2709 file and import it into the destination database.

4.2 Updating the fields of a record in a multi-user environment.

```
(1)  long a;
(2)  long h;
(3)  long r;
(4)  long mfn = 1;
(5)  long times = 100;
(6)  long counter;

(7)  a = IsisAppNew( );
(8)  h = IsisSpaNew(a);
(9)  r = IsisSpaMf(h, “master_name”);

(10) for (counter = 0; counter < times; counter++)  {
(11)          r = IsisRecReadLock(h, 0, mfn);
(12)          if (r == ZERO) {
(13)                  r = IsisRecFieldUpdate(h, 0, “d100  a100#new
                                                              field#”);
(14)                  break;
(15)          } else {
(16)                  seep(1);
(17)          }
(18) }

(19) for (counter = 0; counter < times; counter++)  {
(20)          r = IsisRecWriteUnlock(h, 0);
(21)          if (r == ZERO) {
(22)                  break;
(23)          } else {
(24)                  seep(1);
(25)          }
(26) }

(27) r = IsisAppDelete(a);
```

To update a record in a multi-user environment, it is necessary to lock the desired record before updating it to grant a exclusive write. This is done via IsisRecReadLock function in the line (11). After updating the record it is necessary to write and unlock it, that is done via a call of IsisRecWriteUnlock funstion – line (20). The problem is that it is not granted that we will successed in the lock and unlock of the record (for ex because an exclusive write lock flag set in the database), so it is necessary to check – lines (12) and (21) - the return code and redo the call of the function if it has failed. A good programming style

advises to wait some time (here 1 second) before calling the function again – lines (16) and (24).

4.3 Updating the inverted file from a record in a multi-user environment.

```
(1)  long a;
(2)  long h;
(3)  long r;
(4)  long mfn = 1;
(5)  long times = 100;
(6)  long counter;

(7)  a = IsisAppNew( );
(8)  h = IsisSpaNew(a);
(9)  r = IsisSpaMf(h, “master_name”);
(10) r = IsisSpaIf(h, “inverted_name”);
(11) r = IsisSpaFst(h, “70 0 MHU,(V70/)”);

(12) for (counter = 0; counter < times; counter++)  {
(13)          r = IsisRecReadLock(h, 0, mfn);
(14)          if (r == ZERO) {
(15)                    r = IsisRecFieldUpdate(h, 0, “d70  a70#new field#”);
(16)                    r = IsisRecWrite(h, 0);
(17)                    r = IsisRecIfUpdate(h, mfn);
(18)                    break;
(19)          } else {
(20)                    sleep(1);
(21)          }
(22) }

(23) r = IsisRecUnlock(h, 0);

(24) r = IsisAppDelete(a);
```

This example shows how to update the inverted file in a multi-user environment after the contents of the first record of the database “master_name” are changed.
In the line (11) it is described the field select table (fst) instead of giving a name of a file using the ´@´ symbol. In the line (13) the record is read and locked to avoid the updating of it by another process, in the line (17) the inverted file is updated and finally in the line (23) the record is locked.

4.4 Forcing the unlock of a master file and its records.

```
(1)  long a;
(2)  long h;
(3)  long r;
(4)  long mfn;
(5)  struct IsisRecControl controlStructure;

(6)  a = IsisAppNew( );
(7)  h = IsisSpaNew(a);
(8)  r = IsisSpaMf(h, “master_name”);

(9)  r = IsisSpaMfUnlockForce(h);

(10) r = IsisRecControlMap(h_from, TOCHAR(controlStructure))

(11) for (mfn = 1; mfn < controlStructure.nxtmfn; mfn++)  {
(12)          r = IsisRecRead(h, 0, mfn);
(13)          if (r == ERR_RECLOCKED) {
(14)                  r = IsisRecUnlockForce(h, 0);
(15)          }
(16) }

(17) r = IsisAppDelete(a);
```

The function IsisSpaMfUnlockForce – line (9) – resets the data entry lock (DEL) and the exclusive write lock (EWL) flags of the database, regardless of the ownership or not of the lock by the caller.
In the lines (11) to (16) we read each record of the database base is read (12) and if it is locked, we force the unlock of it – line (14).
The “force” functions should be used with extreme care and only in exceptional situations;