月震データベース ライブラリ仕様
第４版

1993 年 4 月 3 日　　第１版作成
1993 年 8 月 16 日　　第２版作成
1994 年 3 月 31 日　　第３版作成
1997 年 11 月 24 日　　第４版作成

作成者

寺薗　淳也 (文部省宇宙科学研究所　惑星研究系　比較惑星学部門)
荒木　博志 (文部省宇宙科学研究所　惑星研究系　比較惑星学部門)

目 次

Chapter 1

はじめに

本章では、月震データベースへアクセスするためのライブラリ群についての基本的な考え方について述べる。

1.1 ライブラリの基本概念

月震データベースアクセス用ライブラリ (以下 MQLIB) は、月震データベース (以下 MQDB) へのアクセスを行うために開発された、移植性の高い、階層化されたライブラリである。C 及び FORTRAN の両方の言語から利用できるようになっており、幅広いプラットフォームから利用することができる。本ライブラリは、次のような 3 層からなるインタフェース関数、及びユーティリティ関数、FORTRAN 用サブルーチンの合計 5 種類に大別される。

上位インタフェース

C 言語で書かれたユーザプログラムから呼び出され、MQDB データフォーマットで記述されているファイルへのアクセスを行なう。MQLIB では、基本的にユーザプログラム側では上位関数インタフェースのみを呼び出せば、データファイルへのアクセスが可能になるように設計されている。現在 3 つの関数が実装されている。

中位インタフェース

上位インタフェース関数から呼び出され、ヘッダ文字列の操作や、データ部のフォーマット変換、ファイルへの実際のアクセスなどを担当している。現在 46 個の関数が実装されている。

下位インタフェース

上位及び中位インタフェース関数から呼び出され、データ操作の最も基本的な部分を担当する関数である。中位インタフェース関数との差異は、下位インタフェース関数の実行内容は直接 MQLIB とは関係のない基本的な動作に限られている点である。下位インタフェース関数も、ユーザプログラムから呼び出されることは想定していない。現在 9 個の関数が実装されている。

ユーティリティ関数

ファイルへのアクセスとは直接関係しないが、データやファイル操作に役立つ機能を提供する関数である。現在提供されている機能の例を以下に挙げる。

- 平均振幅、最大振幅、サンプリング間隔などの計算あるいは判定
- MQDB 用ファイル名の生成
- 時間の計算

ユーティリティ関数は基本的に、どのインタフェース関数からも呼び出され得るものである。また、ユーザプログラムからも利用できる。現在 14 個の関数が実装されている。

FORTRAN 用サブルーチン

上記、上位インタフェース関数に相当するサブルーチンで、上位インタフェース関数とほぼ同様の機能を FORTRAN から利用できる。FORTRAN のプログラムからは、`call` 命令を利用することによって、通常のサブルーチンと同様にして呼び出すことができる。現在 38 個のサブルーチンが実装されている。

1.2 ライブラリ・バージョン

MQLIB では、ライブラリの将来にわたる変更などに対処するため、ライブラリにはバージョン番号が付けられている。この番号は、次の 3 種類からなる 1 桁の数字の組合せである。

リリース番号

メジャー番号

マイナー番号

ライブラリ・バージョンは、C 言語を利用したプログラム内では随時グローバル変数として参照可能である。また、FORTRAN のプログラム内では、専用のサブルーチンを利用することによって参照することが可能である（未実装)。これらのライブラリ・バージョンの定義方法は、「月震データベース仕様」に述べられている。

1.3 エラーへの対処

MQLIB において発生するエラーは分かりやすい形で処理されると共に、ユーザに迅速に知らされる必要がある。MQLIB ではエラーを次の 3 種類のレベルで分けると共に、エラーメッセージ内にエラーが発生した関数名などを表示することによって、障害原因をいち早く知らせることができるようになっている。

1. 軽度エラー (NOTICE)

   プログラムの実行に影響を与えないエラー。オペレータの介入は必要とせず、プログラムは中断しない。エラーメッセージも出力されない。

2. 中度エラー (WARNING)

   プログラムの実行に影響があるエラー。このエラーの発生時には、MQLIB 関数はデフォルト動作を行なうか、動作を中断して呼び出されたルーチンへ戻る。但しプログラムの実行は中断されない。エラーメッセージが出力される。

3. 重度エラー (FATAL)

   以後のプログラムの実行が不可能な重大なエラー。MQLIB 関数では、エラーが検出されたときにはエラーメッセージを出力して、直ちにプログラムを終了させる。

エラーは現在 71 種類定義されている。また、エラーメッセージの出力のレベルは、ユーザが指定することによって調節できる。例えば、WARNING レベルのエラーであってもエラーメッセージを出力しないようにすることも可能である。

Chapter 2

MQLIB を利用したプログラミング

本節では、MQLIB を利用したプログラミングのための基本的な事項について説明する。

2.1 C 言語

2.1.1 ヘッダファイル

MQLIB で使用される関数や構造体、マクロ定義などの情報は、全てヘッダファイル内に収められている。従ってこれらをプログラムの先頭でインクルードしなければ、MQLIB を利用したプログラムを作成することができない。

MQLIB では、以下に述べる 3 つのインクルードファイルが用意されている。

`mqdb.h`

MQLIB で使用される構造体変数、列挙型変数、マクロの定義や、関数の宣言を行なうヘッダファイルである。

`mqerror.h`

MQLIB におけるエラー番号、エラーメッセージを定義したファイルである。

`mqfunc.h`

MQLIB における関数番号を定義したファイルである。

`mqstrs.h`

MQLIB で使用する文字定数を定義したファイルである。

但し、`mqdb.h` 内で他のファイルをインクルードするため、ユーザプログラムでは `mqdb.h` だけをインクルードすればよい。なお、上記 4 ファイルは、複数回インクルードされてもプログラムのコンパイルに影響はない。

これらのヘッダファイルは、第 B 章で述べるインストールの手順をとった場合には、`/usr/local/include/mqdb` ディレクトリにインストールされる。しかし、通常コンパイラはこのディレクトリをサーチしないので、次のいずれかの方法をとる必要がある。

1. ヘッダファイルのシンボリックリンクを、/usr/include ディレクトリに作成する。但しこの作業は、通常システム管理者のみが実行できる。

2. コンパイル時に、-I オプションを利用して、ヘッダファイルが存在するディレクトリを明示する。通常はこの方法を利用することを薦める。実行例については後述する。

2.1.2 ライブラリ・ファイル

MQLIB 用関数は、通常 libmq.a という名前のライブラリ・ファイルに収められている。このファイルは、第 B 章で述べるインストールの手順をとった場合には、/usr/local/lib ディレクトリにインストールされる。また、XDR ライブラリが実装されていないようなシステムや、ソースパッケージに添付されている XDR ライブラリを使用するようなインストールを行った場合には、libxdr.a というライブラリが同じディレクトリにインストールされる。

ユーザプログラムを作成する際には、リンク時にこれらのライブラリ・ファイルもリンクするようにコンパイラ／リンカに指示する必要がある。このためには、コンパイルまたはリンク時に、-lmq オプションを付加する。また、上で述べた XDR ライブラリを使用する場合には、さらに -lxdr オプションを追加しなくてはならない。システムによっては、システムで用意されている XDR 関連のサポートルーチンが標準の C 言語用ライブラリにない場合がある[1]。このようなシステムでは、さらに-lnsl オプションを付加し、XDR 関係のサポートルーチンをリンクさせる必要がある。

また、/usr/local/lib 以外のディレクトリにファイルをインストールした場合や、リンク時に/usr/local/lib をサーチしないコンパイラ／リンカを使用している場合には、libmq.a が収められているディレクトリを指示するために、-L オプションを使用する。

2.1.3 コンパイル方法

上 2 節で述べた方法を踏まえて、MQLIB を利用したファイルのコンパイルの方法を説明する。まず、第 B 章で述べる標準的なインストールの手順をとった場合における、ユーザプログラム (以下では prog.c というファイル名になっている) のコンパイル例である。

```
cc -o prog prog.c -I/usr/local/include/mqdb -lmq
```

次の例は、ライブラリを/usr/local/lib/mqdb というディレクトリにインストールした場合のコンパイル例である。

```
cc -o prog prog.c -I/usr/local/include/mqdb -L/usr/local/lib/mqdb -lmq
```

2.2 FORTRAN

本章では、ユーザが作成した FORTRAN 言語のプログラムから、MQLIB を利用して MQDB データファイルへアクセスする際の、基本的な的な注意点について述べる。FORTRAN 用サブルーチンについて詳細は第 4 章に述べられている。

[1] System V 系の UNIX システムなど。

2.2.1 コンパイル方法

ここでは、FORTRAN で書かれたプログラムをコンパイルするために必要な事項について説明する。

2.2.2 インクルード・ファイル

FORTRAN プログラムをコンパイルする場合には、C 言語インタフェースを利用する場合のように、特別なファイルをインクルードする必要はない。

2.2.3 ライブラリ・ファイル

FORTRAN 用サブルーチンは、ライブラリ・ファイルの形で提供されている。これらは、`libmqf77.a` という名前のファイルに収められている。従って、このライブラリをリンクすることによって、FORTRAN 用サブルーチンを利用したプログラムのコンパイルが可能になる。しかし、FORTRAN サブルーチンは、その内部で C 言語用の中位／下位インタフェース関数を利用しているため、それらが格納されている `libmq.a` もリンクしなければならない。このため、コンパイル／リンク時には、`-lmqf77 -lmq` オプションを付加する必要がある。

なお、これらのファイルは、第 B 章で述べるインストールの手順をとった場合には、`/usr/local/lib` ディレクトリにインストールされる。

また、`/usr/local/lib` 以外のディレクトリにファイルをインストールした場合や、リンク時に`/usr/local/lib` をサーチしないコンパイラ／リンカを使用している場合には、`libmq.a` が収められているディレクトリを指示するために、`-L` オプションを使用する。

XDR 関連のライブラリについての注意は、2.1.2節に述べられている。

2.2.4 コンパイル方法

上２節で述べた方法を踏まえて、MQLIB を利用したファイルのコンパイルの方法を説明する。まず、第 B 章で述べる標準的なインストールの手順をとった場合における、ユーザプログラム (以下では `prog.f` というファイル名になっている) のコンパイル例である。なお、FORTRAN コンパイル用のプログラム名は処理系によって異なる場合があるので注意すること。ここでは `f77` となっている。

```
f77 -o prog prog.f -lmq -lmqf77
```

次の例は、ライブラリを`/usr/local/lib/mqdb` というディレクトリにインストールした場合のコンパイル例である。

```
f77 -o prog prog.f -L/usr/local/lib/mqdb -lmq -lmqf77
```

2.3 プログラミングにおける注意点

本節では、プログラミングにおける主要な注意点について説明する。

2.3.1 サブルーチン名

FORTRAN 用サブルーチンについては、関数名はすべて 6 文字以内となっており、最初 2 文字は必ず MQ となっている。3 文字目は、

C　　出力用サブルーチン

R　　読み込み用サブルーチン

U　ユーティリティ用サブルーチン

混乱を避けるため、FORTRAN プログラム内では、なるべく MQ で始まるサブルーチン名の利用を避けることが望ましい。

2.3.2 利用するインタフェース関数

MQLIB 関数をプログラムで利用する場合には、必ず上位インタフェース関数を利用するようにする。中位、下位インタフェース関数は、プログラム内で利用するためではなく、上位ライブラリ関数から呼び出されるために定義されていると考えるべきである。ユーティリティ関数は必要に応じて、プログラム内で利用するとよい。

2.3.3 関数名・変数名

MQLIB では、関数及び変数名称は全て Mq... という形で統一されている。従って、Mq... で始まる関数名をユーザ定義の関数につけることは、紛らわしいので極力避けることが望ましい。

2.3.4 マクロ名

マクロ定義においては、EMQ... で始まるマクロ、及び MQ で始まるマクロ名は予約されている。詳しいマクロ名定義については、第 5.4 節に述べられている。

2.3.5 エラーが発生するとき

コンパイルの際にエラーが多発したり、コンパイルは成功したがプログラムがコアダンプするなど、正常に動作しない場合には、次の事項を確認してみるとよい。

1. ライブラリやインクルード・ファイルが存在するディレクトリの指定が誤っていないかどうか確認する。
2. 関数名、変数名、マクロ名に、上節で述べたような、MQ あるいは Mq で始まるような名称を使用していないかどうか調べる。
3. 関数引数の対応関係は正しいか。コアダンプを引き起こすエラーの多くは、引数の対応関係の誤りが原因となっている。特にポインタを渡すべきところで実際の数を渡したり、その逆を行なっていないかどうかチェックする。
4. 処理系は ANSI C に対応しているか。特に、コンパイラは ANSI C 対応でも、ライブラリやヘッダファイルが対応していない場合にはエラーが多発する。
5. ライブラリは正しいものを使用しているか。常に仕様と合致したバージョンのライブラリを使用し、もしバージョンが古くなっていたらインストールをし直す。

Chapter 3

MqRead(), MqCreate() を利用する

本章では、MQDB フォーマットファイルの読み書きを行なう C 言語用上位インタフェース関数である、MqRead(), MqCreate() 両関数の概要を紹介し、併せて両関数を利用したプログラミングについて述べる。両関数の詳しい説明は、第 6 章に述べられている。

3.1 MqRead(), MqCreate() 関数について

本節では、MqRead(), MqCreate() 両関数の概要を紹介する。

3.1.1 MqRead()

MqRead() は、MQDB フォーマットファイルを読み出すための上位インタフェースの 1 つであり、関数を 1 回呼び出すだけでファイル全ヘッダ、及び全データを読み出せるように設計されている。また、MqCreate() は、MQDB フォーマットファイルを作成するための上位インタフェースであり、同じく関数を 1 回呼び出すことにより、全ヘッダ、及び全データを作成することが可能である。またヘッダのアクセスについては、アクセス制御変数を利用することによる調節が可能で、不必要なヘッダを読み書きしないように設定できる。

本関数を利用すると、次章で述べる方法、すなわち、各ヘッダへのアクセス用関数をそのたび毎に呼び出す方法に比べて、コーディングが大変簡単になるほか、標準入出力の扱いなどが比較的楽である。

3.2 データファイルの作成

MQLIB 用データファイルを作成する場合には、基本的に次のような手順を踏む。

1. プログラムのインクルードファイルに、mqdb.h を追加する。このファイルをインクルードしないと、MQLIB 関数や変数の定義などが使用できない。
2. まず、MqAccess 構造体変数を定義する。この変数は、ヘッダ作成関数が作成するヘッダを規定する。
3. ヘッダのデータを格納する MqHeader 構造体変数を用意する。
4. 実際のデータの「受け皿」となる double *型変数を用意する。この変数には実際に出力されるデータを格納する。データの格納の順序であるが、チャンネル 1 のデータは配列の $0 \sim n-1$ 番目、チャンネル 2 のデータは $n \sim 2n-1$ 番目といったように格納しておく。

5. MQLIB の関数を使用する前に、必ず MqInit() 関数を呼ぶ。この関数は、状況変数 (第 5.6 節参照) を初期化し、ライブラリを利用できる状態にするために必ず必要となるものである。

6. ファイル名を格納した char *型変数を用意する。この変数にはファイル名を格納しておいてもよいし、もし標準出力を利用してデータを出力する場合には、NULL をセットしておいてもよい。

7. MqAccess 構造体変数を使って、アクセスしたいヘッダを選ぶ。具体的には、構造体の各メンバに、アクセスしたい場合には MQ_YES を、アクセスしたくない場合には MQ_NO を代入すればよい。すべてのヘッダにアクセスしたい場合には、ユーティリティ関数の MqSetAccessYes() を利用すれば、全メンバに MQ_YES がセットされる。また、ヘッダ 1 つだけにアクセスしたい場合には、まず、全メンバに MQ_NO をセットするユーティリティ関数 MqSetAccessNo() を実行し、その後アクセスしたいメンバだけに MQ_YES を代入すればよい。

8. MqCreate() を呼び出す。この関数は、引き渡された MqAccess 構造体変数の指定に従って、MqHeader 構造体変数に格納されたデータを使ってヘッダを作成し、その後データ部を作成する。データファイルの作成が成功すれば、エラー値 (グローバル変数である MqErr.errno) は EMQNOERROR である。もし作成途上で何らかのエラーが発生した場合には、この変数に然るべき値が代入される。エラーの種類によって、エラーメッセージが出力されるだけでデータファイルの作成が続けられる場合と、データファイルの作成が異常終了する場合がある。詳しくは第 5.4 節を参照のこと。

以下は、MQDB フォーマットファイルを出力するプログラムの例である。基本的な構造だけを示してある。

```
#include<stdio.h>
#include<mqdb.h>              /* 必ずこのファイルをインクルードすること */

main()
    {
    MqAccess    access;       /* アクセス制御用変数 */
    MqHeader    header;       /* ヘッダデータ格納用変数 */
    char        filename[] = "test.dat"; /* データファイル名 */
    double     *data;         /* 実際のデータを格納するための変数 */

    MqInit( "prog-1.00" );          /* ライブラリ初期化 */
    MqSetAccessYes( &access );      /* すべてのヘッダを作成する */

        :
        :  この部分で、MqHeader 変数や double *変数へのデータの格納を行なう。
        :

    MqCreate( filename, &access, &header, data )

    if( MqErr.errno == EMQNOERROR )
        exit( EXIT_SUCCESS );
    else
        exit( EXIT_FAILURE );
    }
```

3.3 データファイルの読み込み

MQLIB 用データファイルを読み込むプログラムを作る場合には、次のようなプログラムを書く。

1. プログラムのインクルードファイルに、mqdb.h を追加する。このファイルをインクルードしないと、MQLIB 関数や変数の定義などが使用できない。

2. まず、MqAccess 構造体変数を定義する。この変数は、読み込み関数においてアクセスするヘッダを規定する。

3. ヘッダのデータを格納する MqHeader 構造体変数を用意する。

4. 実際のデータの「受け皿」となる double *型変数を用意する。この変数は MqRead() から返されるデータ配列のポインタを受けとるために必ず用意されなくてはならない。

5. プログラム初頭に、必ず MqInit() 関数を呼ぶ。この関数は、状況変数 (第 5.6 節参照) を初期化し、ライブラリを利用できる状態にするために必ず必要となるものである。

6. ファイル名を格納した char *型変数を用意する。この変数にはファイル名を格納しておいてもよいし、もし標準入力を利用してデータを受けとる場合には、NULL をセットしておいてもよい。

7. MqAccess 構造体変数を使って、アクセスしたいヘッダを選ぶ。具体的には、構造体の各メンバに、アクセスしたい場合には MQ_YES を、アクセスしたくない場合には MQ_NO を代入すればよい。すべてのヘッダにアクセスしたい場合には、ユーティリティ関数の MqSetAccessYes() を利用すれば、全メンバに MQ_YES がセットされる。また、ヘッダ 1 つだけにアクセスしたい場合には、まず、全メンバに MQ_NO をセットするユーティリティ関数 MqSetAccessNo() を実行し、その後アクセスしたいメンバだけに MQ_YES を代入すればよい。

8. MqRead() 関数を呼び出す。この関数は、引き渡された MqAccess 構造体変数の指定に従って、ヘッダのデータを MqHeader 構造体変数に読み込む。もしデータの読み込みが成功すれば、データの先頭のポインタを返す。もしデータの読み込みに失敗した場合には、NULL が返される。

   データは、チャンネルの順番に格納される。すなわち、チャンネル 1 のデータは配列の $0 \sim n-1$ 番めまでに格納される。チャンネル 2 のデータは配列の $n \sim 2n-1$ 番めまでに格納されている。

データを得る操作はこれで終りである。あとは必要に応じて、変数を別の処理用ルーチンに引き渡したり、ヘッダの内容に応じた処理を行なえばよい。

以下に、MQDB フォーマットファイルを読み込むプログラムの例を示す。このプログラムは、test.dat というファイルからデータを読み込み、その最初のデータの内容を表示する。

```
#include <stdio.h>
#include <mqdb.h>              /* 必ずこのファイルをインクルードすること */

main()
    {
    MqAccess    access;        /* アクセス制御用変数 */
    MqHeader    header;        /* ヘッダデータ格納用変数 */
    char        filename[] = "test.dat"; /* データファイル名 */
    double     *data;          /* 実際のデータを格納するための変数 */
```

```
    MqInit( "prog-1.00" );          /* ライブラリ初期化 */
    MqSetAccessYes( &access );      /* すべてのヘッダにアクセスする */

    /* もしデータの読み込みに失敗したらメッセージを出力 */
    if( NULL == ( data = MqRead( filename, &access, &header )))
MqPerror( NULL );

    printf( "%lf\n", data[0] );      /* データの最初を出力 */
    }
```

Chapter 4

FORTRAN 用サブルーチンを利用する

4.1 FORTRAN サブルーチンの概要

FORTRAN 用インタフェースを用いて MQDB データファイルにアクセスする場合は、第 3 章で述べた、MqCreate(), MqRead() を利用する場合とは異なり、ファイル記述子 (file descriptor) [1]によるファイル管理を行なう必要がある。すなわち、次のような手順を踏まなければならない。

1. MQLIB の初期化。これは MQINIT() サブルーチンを呼ぶことによって行なわれる。これはプログラム内で、MQLIB サブルーチンを利用する前に 1 回だけ行なえばよい。
2. ファイルのオープン。これは MQOPEN() サブルーチンにより行なわれる。このサブルーチンを呼ぶと、MQLIB はファイル記述子（実際には 0 以上の整数である）をプログラムに返す。以後、ファイルへのアクセスはこのファイル記述子によって行なわれる。
3. ファイルへのアクセス。
4. ファイルのクローズ。これは MQCLOS() サブルーチンにより行なわれる。このサブルーチンを呼ぶことによって、利用していたファイル記述子は破棄される。

このうち、2 〜 4 の段階は、プログラム中で何回も繰り返してよい。但し、同時に利用できるファイル記述子の数は 16 個までである。従って、多数の MQDB フォーマットファイルをオープンする場合には、必ず使用後に MQCLOS() サブルーチンを呼び、ファイル記述子を破棄するようにする。さもないと、ファイル記述子が不足してしまう可能性がある。

4.2 データファイルの作成

FORTRAN を利用したデータファイルの作成は、次のようにして行なう。

1. 書き出したい変数を用意する。変数の型に注意すること。また、必ずプログラム署名を用意しておく。
2. MQINIT() サブルーチンを呼び出す。これによって、MQLIB の変数が初期化され、サブルーチンが利用できる状態となる。ここでプログラム署名を引数で渡す。

[1]ここで述べる「ファイル記述子」は、UNIX システムなどで使用される「ファイル記述子」とは（似てはいるものの）全く異なるものである。

3. MQOPEN() サブルーチンを呼び出す。このサブルーチンは、次のような仕様となっている。

```
MQOPEN( CHARACTER FILENAME, INTEGER*2 MQRW, INTEGER*2 MQFILE,
INTEGER FILDESC )
```

ここで、各引数は次のような意味を持つ。

引数	意 味
FILENAME	ファイル名。もし標準入出力または標準エラー出力に対して入出力を行なうのであれば、この変数は空白文字で構わない。入出力先は第３引数で指定する。
MQRW	入力 (読み) か出力 (書き) かの指定。ファイルを読み出す場合には 0、ファイルに書き出す場合には 1 を指定する。**それ以外の数値は許されない。**
MQFILE	入出力先。通常のファイルへの入出力の場合には 0、標準入力へアクセスする場合には 1、標準出力にアクセスする場合には 2、標準エラー出力にアクセスする場合には 3 を指定する。**それ以外の数値は許されない。**
FILDESC	ファイル記述子。以後サブルーチンのアクセスには、ここで得られたファイル記述子が使用される。

ファイルへの書き出しを行う場合には、MQRW を 1、MQFILE を 0 にセットし、MQOPEN() を呼び出す。この呼び出しによって、変数 FILDESC にファイル記述子がセットされる。もしエラーが発生すれば、プログラムが異常終了する。標準出力へ書き出す場合には、MQFILE に 2 を指定する[2]。

4. ヘッダ書き出し用のサブルーチンを呼び出す。サブルーチンを呼び出す場合には、第１引数には必ず、MQOPEN() により得られたファイル記述子を指定すること。決してファイル名を指定してはならない。

5. データ書き出し用サブルーチン MQCD() を呼び出す。このサブルーチンは、データを自動的に書き出す一方、ヘッダデータも指定されたファイルへ出力する。すなわち、MQCD() の呼び出しによってデータファイルが作成される。

6. MQCLOS() サブルーチンを呼び出す。これによって、利用していたファイル記述子が破棄され、再利用できる状態となる[3]。

以下の例は、MQDB フォーマットファイルを書き出すプログラムである。

```
CHARACTER   INFILE*80, OUTFILE*80, PROGSIGN*16
INTEGER*2   CHANNELS, DATAFORM, STATION, TAPENUM, OBSMODE
INTEGER*2   FDIN, FDOUT
INTEGER*2   DATATYPE, FILETYPE
INTEGER*2   AACHANNEL, MXCHANNEL, SRCHANNEL
INTEGER*2   YEAR, DAY, HOUR, MINUTE, SECOND, MS
INTEGER*4   STARTREC, ENDREC, NOD
REAL*8      AVEAMP(255), MAXAMP(255), SAMPRATE(255)
REAL*8      DATA(100000)

INTEGER*2   MQREAD, MQWRITE
```

[2]標準エラー出力に書き出すことも可能であるが、もともとエラー出力を行なうために設けられている出力先であるから、利用しない方が望ましい。

[3]MQCLOS() サブルーチンの呼び出しによって、該当するファイル記述しについての全ての情報が消去される。従って、MQCLOS() サブルーチンを呼び出す前に、必ず MQCD() サブルーチンが呼び出されていることを確認するべきである。

```
INTEGER*2   MQFILE, MQSTDIN, MQSTDOUT, MQSTDERR

MQREAD   = 0
MQWRITE  = 1

MQFILE   = 1
MQSTDIN  = 2
MQSTDOUT = 3
MQSTDERR = 4

OUTFILE = 'test.lp'
PROGSIGN = 'ftest'

CALL MQINIT( PROGSIGN )

CALL MQOPEN( OUTFILE, MQWRITE, MQFILE, FDOUT )

CALL MQCAA( FDOUT, AACHANNEL, AVEAMP )
CALL MQCCH( FDOUT, CHANNELS )
CALL MQCDF( FDOUT, DATAFORM )
CALL MQCDT( FDOUT, DATATYPE )
CALL MQCERC( FDOUT, ENDREC )
CALL MQCFT( FDOUT, FILETYPE )
CALL MQCMA( FDOUT, MXCHANNEL, MAXAMP )
CALL MQCNOD( FDOUT, NOD )
CALL MQCOM( FDOUT, OBSMODE )
CALL MQCSTN( FDOUT, STATION )
CALL MQCSR( FDOUT, SRCHANNEL, SAMPRATE )
CALL MQCSRC( FDOUT, STARTREC )
CALL MQCST( FDOUT, YEAR, DAY, HOUR, MINUTE, SECOND, MS )
CALL MQCTN( FDOUT, TAPENUM )

CALL MQCD( FDOUT, DATA )

CALL MQCLOS( FDOUT )
STOP
END
```

4.3 データファイルの読み込み

FORTRAN を利用してデータファイルの読み込みを行なうプログラムを書く際には、次のような手順を踏む。

1. 必要な変数を用意する。

2. MQINIT() サブルーチンを呼び出す。これによって、MQLIB の変数が初期化され、サブルーチンが利用できる状態となる。

3. MQOPEN() サブルーチンを呼び出す。このサブルーチンの仕様は前節に述べられている。ファイルの読み込みを行う場合には、MQRW を 0、MQFILE を 0 にセットし、MQOPEN() を呼び出す。この呼び出しによって、変数 FILDESC にファイル記述子がセットされる。もしエラーが発生すれば、プログラムが異常終了する。標準入力から MQDB フォーマットファイルを読み出す場合には、MQFILE に 1 をセットする。

4. ヘッダ、データ読み出し用のサブルーチンを呼び出す。サブルーチンを呼び出す場合には、第 1 引数には必ず、MQOPEN() により得られたファイル記述子を指定すること。決してファイル名を指定してはならない。

5. MQCLOS() サブルーチンを呼び出す。これによって、利用していたファイル記述子が破棄され、再利用できる状態となる。

以下の例は、MQDB フォーマットファイルを読み込むプログラムの例である。この例では deepmq.lp という名前の MQDB フォーマットファイルを読み込んでいる。

```
CHARACTER   INFILE*80, PROGSIGN*16
INTEGER*2   CHANNELS, DATAFORM, STATION, TAPENUM, OBSMODE
INTEGER*2   FDIN, FDOUT
INTEGER*2   DATATYPE, FILETYPE
INTEGER*2   AACHANNEL, MXCHANNEL, SRCHANNEL
INTEGER*2   YEAR, DAY, HOUR, MINUTE, SECOND, MS
INTEGER*4   STARTREC, ENDREC, NOD
REAL*8      AVEAMP(255), MAXAMP(255), SAMPRATE(255)
REAL*8      DATA(100000)

INTEGER*2   MQREAD
INTEGER*2   MQFILE, MQSTDIN, MQSTDOUT, MQSTDERR

MQREAD   = 0
MQWRITE  = 1

MQFILE   = 1
MQSTDIN  = 2
MQSTDOUT = 3
MQSTDERR = 4

INFILE = 'deepmq.lp'
PROGSIGN = 'ftest'

CALL MQINIT( PROGSIGN )

CALL MQOPEN( INFILE, MQREAD, MQFILE, FDIN )
```

```
CALL MQRAA( FDIN, AACHANNEL, AVEAMP )
CALL MQRCH( FDIN, CHANNELS )
CALL MQRDF( FDIN, DATAFORM )
CALL MQRDT( FDIN, DATATYPE )
CALL MQRERC( FDIN, ENDREC )
CALL MQRFT( FDIN, FILETYPE )
CALL MQRMA( FDIN, MXCHANNEL, MAXAMP )
CALL MQRNOD( FDIN, NOD )
CALL MQROM( FDIN, OBSMODE )
CALL MQRSTN( FDIN, STATION )
CALL MQRSR( FDIN, SRCHANNEL, SAMPRATE )
CALL MQRSRC( FDIN, STARTREC )
CALL MQRST( FDIN, YEAR, DAY, HOUR, MINUTE, SECOND, MS )
CALL MQRTN( FDIN, TAPENUM )
CALL MQRD( FDIN, CHANNELS, NOD, FILETYPE, DATA )

CALL MQCLOS( FDIN )

STOP
END
```

Chapter 5

変数・構造体・マクロ定義

5.1 定数定義

MQLIB に関係する全ての定数は、ファイル mqstrs.h で定義されている。このファイルは mqdb.h でインクルードされているので、改めてインクルードする必要はない。また定数のうち、エラー番号関連のものは mqerror.h で、関数番号及び関数名については mqfunc.h で定義されている (第 5.5 節参照)。

mqstrs.h で定義されている定数については、巻末の第 D 章に一覧表がある。

5.2 列挙型変数定義

現在定義されている列挙型変数は次の通りである。

5.2.1 MQAM

MQAM は、ヘッダにアクセスする際に、読み込んだ、あるいはファイルに書き込むためのヘッダ情報を一時的に蓄える方法を指定する変数である。但し、MqRead() 及び MqCreate() によるアクセスの場合には、この変数による制御は適用されない。

```
typedef enum { MQ_DIRECT, MQ_TMP, MQ_SHM, MQ_EMS } MQAM;
```

各変数の意味は次の通りである。

MQ_DIRECT データファイルに直接アクセスしてヘッダの読み書きを行なう。これは標準入出力に対しては無効であり、エラーとなる。

MQ_TMP ヘッダの内容を一時ファイルに書き出した上で、アクセスする。ファイルの読み出しの場合には、読み出されたヘッダの内容が一時ファイルに書き出され、ファイルの書き出しの場合には、書き出したいヘッダ情報が一時ファイルに書き出される。一時ファイルが作成されるディレクトリは /tmp である。一時ファイルの名称は、次のようになる。

mqdb.[プロセス番号].[ファイル記述子]

ここで、プロセス番号は現在実行中のプログラムのプロセス番号であり、ファイル記述しはアクセスしているファイルに適用されているファイル記述子である。なお、同名のファイルが存在した場合

には、強制的に上書きされる[1]。このファイルは、MqOpen() 関数 (C 言語) または MQOPEN() サブルーチン (FORTRAN) が呼び出された時に自動的に作成され、MqClose() (C 言語) または MQCLOS() (FORTRAN) が呼び出された時に自動的に削除される[2]。

MQ_SHM 未実装 ヘッダの内容を共有メモリに蓄積する。

MQ_EMS 未実装 ヘッダの内容を EMS メモリに蓄積する。MS-DOS 以外では利用できない。

MqInit() 関数 (サブルーチン) を呼びだし、MQLIB を初期化した直後は、ファイルアクセス方法は MQ_TMP となる。

本変数は、MqFile 構造体内で使用される (第 5.3.3 節参照)。

5.2.2 MQDF

MQDF は、月震の元ファイルにおけるデータのフォーマットの種類を示す。

```
typedef enum { MQ_OLD, MQ_NEW } MQDF;
```

本変数において、MQ_OLD は元ファイルのデータフォーマットが OLD フォーマットであることを、MQ_NEW は NEW フォーマットであることを示す。なお、アポロ月震データのフォーマットについては、寺薗 (1992) などに詳述されている。

5.2.3 MQDT

MQDT は、月震のデータの種別を表すための列挙型変数である。

```
typedef enum { MQ_LP, MQ_SP, MQ_TIDAL }  MQDT;
```

本変数において、MQ_LP は長周期データを、MQ_SP は短周期データを、MQ_TIDAL は潮汐データを表す。

5.2.4 MQFT

MQFT は、MQDB フォーマットファイルのフォーマットの種類を示す。

```
typedef enum { MQ_FULLTEXT, MQ_COMPOSITE, MQ_XDR }    MQFT;
```

MQ_FULLTEXT は、ヘッダ部、データ部共にテキストデータであることを示し、MQ_COMPOSITE は、ヘッダ部のみがテキストで、データ部はバイナリ出力されていることを示す。なお、このバイナリ出力部は機種、処理系及び CPU のバイトオーダに依存するため、MQ_COMPOSITE ファイルについては、異機種間、あるいは異なる処理系でコンパイルされたライブラリ間でのデータの互換性は保証されない。MQ_XDR は、XDR ライブラリを利用したデータ表現形式によるフォーマットである。XDR ライブラリを使用した場合には、データ部はバイナリ出力であるが、異機種間、あるいは異なる処理系でコンパイルされたライブラリ間でのデータ互換性は保証される。なお、MQDB におけるデータ表現形式について詳しくは、「月震データベース仕様書」に述べられている。

---

[1] ファイル名の性質上、同名のファイルが存在する可能性はほぼないと考えてよい。但し、/tmp を数台のコンピュータ間で共用している場合などには、別のプログラムで使用していた一時ファイルが上書きされることがあり得るので、注意する必要がある。

[2] 但し、プログラムが外部から強制終了された場合や、内部エラーなどにより異常終了した場合には、一時ファイルが残ったままとなる。

5.2.5 MQIO

MQIO は、アクセスするファイルの種類を指定する。MqRead() 及び MqCreate 関数を利用してファイルにアクセスする場合には、本変数の指定は意味を持たない。

```
typedef enum { MQ_FILE, MQ_STDIN, MQ_STDOUT, MQ_STDERR } MQIO;
```

各変数の意味は次の通り。

MQ_FILE 通常のファイルに対して入出力を行なう。

MQ_STDIN 標準入力から読み込む。

MQ_STDOUT 標準出力へ書き出す。

MQ_STDERR 標準エラー出力へ書き出す。なお、標準エラー出力は本来、エラーメッセージの出力のための用意されているものであって、データの出力先として利用することは好ましくない。端末へのデータ出力などでは、なるべく MQ_STDOUT を利用する。

本変数は、MqFile 構造体内で使用される (第 5.3.3 節参照)。MqOpen() または MQOPEN() 関数／サブルーチンにおいて、MQRW 変数 (第 5.2.7 節参照) に MQ_WRITE がセットされている際に、MQIO 変数に MQ_STDIN を指定するとエラーとなる。同様に、MQRW 変数に MQ_READ がセットされされている状況で、MQIO 変数に MQ_STDOUT を指定してもエラーとなる。

5.2.6 MqMode

MqMode は、月震の観測モードを表すための列挙型変数である。

```
typedef enum { MQ_PEAKED, MQ_FLAT }      MqMode;
```

本変数において、MQ_PEAKED は peaked モードでの観測、MQ_FLAT は flat モードでの観測が行なわれたことを示す。

5.2.7 MQRW

MQRW は、MQDB フォーマットファイルへのアクセスの形態を指定する。MqRead() 及び MqCreate 関数を利用してファイルにアクセスする場合には、本変数の指定は意味を持たない。

```
typedef enum { MQ_READ, MQ_WRITE, MQ_UNKNOWN } MQRW;
```

MQ_READ は読み出しを、MQ_WRITE は書き出しを意味する。MQ_UNKNOWN はアクセス形態が未決定であることを意味するため、MQLIB 内でこの状態のままファイルアクセスが行なわれるとエラーとなる。
本変数は、MqFile 構造体内で使用される (第 5.3.3 節参照)。

5.2.8 MqStat

この列挙型変数は、ヘッダへのアクセスの成功／失敗を記録するために、状況変数 MqStatus で利用される (第 5.6 節参照)。

```
typedef enum { MQ_NOACCESS, MQ_SUCCESS, MQ_FAIL } MqStat;
```

MQ_NOACCESS は、該当するヘッダのアクセスが行なわれなかったことを示し、MQ_SUCCESS は、アクセスが成功した時にセットされる。MQ_FAIL は、アクセスが失敗した際の値である。

5.2.9 MqYesNo

MqYesNo は、ヘッダの読み書きなどにおいてアクセスするヘッダを特定するために用いられる (第 5.3.1 節参照)。また一般的に、yes 及び no を表す Boolean 変数としての利用も可能である。

```
typedef enum { MQ_YES, MQ_NO } MqYesNo;
```

5.3 構造体定義

MQLIB において現在定義されているデータ構造体は次の通りである。

5.3.1 MqAccess

MqAccess は、月震データファイルのヘッダへのアクセスを制御するための構造体であり、主に MqRead(), MqCreate() 関数を利用したファイルアクセスの際に、アクセスするヘッダを指定するために用いられる。

ヘッダを読みとる際に MqRead() 関数を利用する場合には、この構造体変数に値 MQ_YES がセットされているヘッダだけが読みとられる。他のヘッダについては読みとりが行なわれない。

ヘッダを作成する際に MqCreate() 関数を利用する場合には、この構造体変数に MQ_YES がセットされているヘッダだけが作成される。他のヘッダは作成されない。

```
typedef struct {
    MqYesNo  AveAmp;        /* 平均振幅 */
    MqYesNo  Channels;      /* チャンネル数 */
    MqYesNo  DataCreDate;   /* データ作成日時 */
    MqYesNo  DataFormat;    /* 元ファイルのデータフォーマット */
    MqYesNo  DataModDate;   /* データの更新日時 */
    MqYesNo  DataType;      /* データ種別 ( OLD/NEW ) */
    MqYesNo  EndRecord;     /* 元ファイルのデータ終了レコード数 */
    MqYesNo  MaxAmp;        /* 最大振幅 */
    MqYesNo  NumOfData;     /* データ数 */
    MqYesNo  ObsMode;       /* 観測モード ( PEAKED / FLAT ) */
    MqYesNo  OrgFileName;   /* 元データのファイル名 */
    MqYesNo  SampRate;      /* サンプリング間隔 */
    MqYesNo  Station;       /* 観測ステーション */
    MqYesNo  StartRecord;   /* 元ファイルのデータ開始レコード数 */
    MqYesNo  StartTime;     /* データの開始時刻 */
    MqYesNo  TapeNumber;    /* テープ番号 */
    }    MqAccess;
```

5.3.2 MqAmp

MqAmp は、最大振幅やサンプリング間隔など、チャンネルによって値が異なり得る情報を格納するための構造体である。チャンネルによって値が異なっている場合には、メンバ channels には 1 以上の数値が格納される。すべてのチャンネルで同じ値をとるときには、channels には 1 がセットされる。

```
typedef struct {
```

```
    unsigned short  channels;          /* データのチャンネル数 */
    double     data[MQCHANNELMAX+1];   /* データ */
    }   MqAmp;
```

5.3.3 MqFile

MqFile は、MQLIB においてファイル入出力の管理のために用いられる (MqRead() 及び MqCreate() 関数を利用した場合を除く)。

```
typedef struct {
    MqYesNo   InUse;       /* ファイル記述子が使用されているかどうかのフラグ */
    MQAM      accmethod;   /* アクセス方法 */
    MQIO      iostream;    /* 入出力の方法 (ファイル、標準入出力)  */
    MQRW      ReadWrite;   /* 入出力の方向 (入力か出力か)  */
    char     *filename;    /* 実際のファイル名 */
    MqLibInfo libinfo;     /* 読みとられたライブラリ情報 */
```

本変数は、プログラム内ではグローバル変数 MqFilDesc として使用される。

5.3.4 MqHeader

MqHeader 構造体は、ヘッダのデータをやりとりするための情報を格納する構造体である。この構造体は MqRead() 及び MqCreate() 関数内で、ヘッダのデータを構造体単位で授受するために使われる。

```
typedef struct MqHeader    {
    MqAmp           AveAmp;       /* 平均振幅 */
    unsigned short  Channels;     /* チャンネル数 */
    MQDF            DataFormat;   /* データのフォーマット */
    MQDT            DataType;     /* データ種別 */
    unsigned long   EndRecord;    /* 元ファイルにおけるデータ終了レコード数 */
    MQFT            FileType;     /* データファイル種別 */
    MqAmp           MaxAmp;       /* 最大振幅 */
    time_t          ModDate;      /* 修正日時 */
    unsigned long   NumOfData;    /* データ数 */
    MqMode          ObsMode;      /* 観測モード */
    char            OrgFile[FILENAME_MAX];  /* 元のデータファイル名 */
    MqAmp           SampRate;     /* サンプリング間隔 */
    unsigned long   StartRecord;  /* 元ファイルにおけるデータ開始レコード数 */
    MqTime          StartTime;    /* データの開始時刻 */
    unsigned short  Station;      /* 観測ステーション */
    unsigned short  TapeNumber;   /* テープ番号 */
    }    MqHeader;
```

個々の変数名はヘッダに対応する。変数の内容については、「月震データベース仕様」に述べられている。

5.3.5 MqLibVer

MqLibVer 構造体は、MQDB フォーマットで記述されたデータファイルの先頭行に記されている、ライブラリについての情報を受渡しするための構造体である。この構造体は、以下のようになっている。

```
typedef struct {
    char  LibType[2];        /* ライブラリ種類識別子 */
    char  Release;           /* リリース番号 */
    char  Major;             /* メジャー番号 */
    char  Minor;             /* マイナー番号 */
    }   MqLibVer;
```

LibType には、英文字２文字で記されるライブラリ種類識別子が入る。Release、major、minor には、それぞれ、ライブラリ番号、メジャー番号、マイナー番号が格納される。但し、それぞれの変数の型は char 型であり、数字ではありながら short や long などではない。

同様に、LibType もライブラリ識別子の英２文字分しか領域が用意されておらず、通常の C 言語における文字列のように、最後が \0 とはなっていない。従って、LibType から通常の文字列変数などにコピーする際、あるいはその逆の操作を行なう際、strcpy() などの通常の文字列関数を利用するとメモリエラーを引き起こす可能性がある。strncpy() 関数などを利用するか、直接文字列を代入するようにプログラムを組む必要がある。

5.3.6 MqLibInfo

MqLibInfo は、MQDB フォーマットファイル先頭行に記されている情報全てを受渡しするための構造体である。形は以下のようになっており、上記 MqLibVer にプログラム署名の情報が加わっただけである。本構造体を利用する際の注意点などは、上記 MqLibInfo の場合と基本的に同じである。

```
typedef struct {
    MqLibVer    LibVer;                  /* ライブラリのバージョン */
    char        ProgSign[MQPROGSIGNLEN];  /* プログラム署名 */
    }   MqLibInfo;
```

5.3.7 MqTime

MqTime は、主に月震のデータ開始時刻を格納するために利用される構造体であり、次のような形式となっている。

```
typedef struct {
    unsigned short    year;     /* 年 */
    unsigned short    day;      /* 日 ( 1 から 366 ) */
    unsigned short    hour;     /* 時 ( 0 から 23 ) */
    unsigned short    minute;   /* 分 ( 0 から 59 ) */
    unsigned short    second;   /* 秒 ( 0 から 59 ) */
    unsigned short    ms;       /* マイクロ秒 ( 0 から 999 ) */
    }  MqTime;
```

5.4 マクロ定義

MQLIB としての独自のマクロ定義は比較的少なく、下に挙げてあるだけである。但し、エラー番号及び関数の番号は全てマクロによって定義されている。これについては、巻末付録第 C 章に全て表として掲載されている。

```
#define MQFILENAMELEN    12  /* ファイル名の長さ */
#define MQFILDESCMAX     16  /* 使用できるファイル記述子の数 */
#define MQLIBVERLEN       5  /* ライブラリ番号の最大長 */
#define MQPROGSIGNLEN    16  /* プログラム署名の最大の長さ (バイト) */
#define MQHDRLENMAX    1024  /* ヘッダの最大の長さ (バイト) */
#define MQHDRLINEMAX     80  /* ヘッダにおける 1 行の長さの最大値 */
#define MQCHANNELMAX    255  /* 最大チャンネル数 */
#define MQTOPLINEMIN     12  /* 先頭行の長さの最小値 (バイト) */
```

また、MQLIB のライブラリ種別、リリース番号、メジャー番号、マイナー番号も、マクロで定義されている。

```
#define MQLIB_LIBTYPE   ``MQ''  /* ライブラリ種別 */
#define MQLIB_RELEASE     1     /* リリース番号 */
#define MQLIB_MAJOR       0     /* メジャー番号 */
#define MQLIB_MINOR       0     /* マイナー番号 */
```

その他に、次のようなマクロ定義が mqdb.h 内に記述されている。これらは直接 MQLIB の動作には関係しないが、ユーザ・プログラム内で利用することもできる。このうち FILENAME_MAX は、本来 ANSI C 対応の処理系であれば stdio.h などで定義されているはずであり、MQLIB では未定義の場合のみ、改めて定義し直している。従って、処理系で既に FILENAME_MAX が定義されている場合には、そちらが優先される。

```
#define MQ_TRUE        1   /* Boolean 型変数 */
#define MQ_FALSE       0   /* Boolean 型変数 */
#define FILENAME_MAX 255   /* ファイル名の最大長 */
```

5.5 エラー番号・関数番号定義

エラー番号及びエラーレベルは、全てエラー関連の変数・マクロを定義しているファイル mqerror.h に定義されている。このヘッダファイルは、ライブラリ作成時に、エラー番号、エラーレベル (後述) 及びそれに対応するエラーメッセージを記述したファイル mqerror.list 、及び、元となるファイル mqerror.tmpl から、自動的に生成される。

また、エラーが発生した場合、その関数及びサブルーチンの名前が自動的に番号でセットされる。この関数の番号を「関数番号」といい、これらは mqfunc.h で定義されている。MQLIB 内でエラーが発生すると、mqerror.h 内で定義されているグローバル変数 MqErr に値がセットされる。この変数は次のような構造体となっており、各メンバーは次のような意味を持つ。

```
struct {
    unsigned long  errno;  /* エラー番号 */
```

```
unsigned long  func;   /* エラーが発生した関数の番号  */
unsigned short level;  /* エラーのレベル */
unsigned short warn;   /* エラーメッセージを出力するエラーのレベル */
}  MqErr;
```

エラー番号については後述する。

MQLIB では、エラーに対し、エラー番号と、そのエラーの重大度を表すエラーレベルの 2 種類の情報が定義される。エラーのレベルは各エラーについて一意に決められており、エラーのレベルによって処理が異なる (第 1.1 節参照)。

関数番号は、該当する関数・サブルーチン名を全て大文字に直したマクロ名で定義されている。例えば、MqCreate の関数番号のマクロ名は MQCREATE である。実際の関数名は、グローバル変数 **MqFuncName に定義されており、配列の番号にマクロ名を指定すれば、実際の関数名が入った文字列の先頭ポインタが得られる。例えば、次の printf 文は (大変つまらない例ではあるが)、MQCREATE で示される関数の名前 (当然 MqCreate である) を出力する。

```
printf( "%s\n", MqFuncName[MQCREATE] );
```

MQLIB の関数は、最初に必ず関数番号をセットするようになっている。そのため、エラーが起きた場合、必ずその関数番号が参照される。但し、MQLIB 関数がさらに上の MQLIB 関数から呼び出されている場合などは、必ずしもエラーの原因が該当する関数ではない場合もあるので、デバッグなどの際には注意する必要がある。

構造体の中で、warn はユーザが直接操作することが可能である。このメンバは、エラーメッセージをどのレベルのエラーが発生した場合に出力するかを制御する。MQLIB では、エラーメッセージは本メンバ以上のレベルのエラーが発生した場合に出力される。例えば、warn が EMQ_FATAL となっている場合には、エラーレベルが FATAL の場合にのみエラーが出力される。

warn の制御は、本メンバに直接変数を代入することによっても可能であるが、間違いを防ぐために、なるべくユーティリティ関数 MqSetWarnLevel() を利用するようにする。

プログラム中では、エラー番号やエラーが発生した関数の番号などを自由に参照できる。しかし、将来にわたってこれらの番号は変更され得るので、エラー番号、関数番号とも、マクロ名を使用することが望ましい。全エラー番号とその意味については、巻末の第 C 章に一覧表がある。

5.6 状況変数

MQLIB 内では、ヘッダの読み込みの成功／失敗や、ライブラリのバージョンなどを、定数として保持してあり、必要に応じてユーザプログラムから参照することができる。これらの変数は**状況変数** (condition variable) という。上節で述べたエラー番号もこのような変数の一種と考えることができるが、MQLIB 内では、MqErr 構造体の他に、次の 3 つの構造体が状況変数として定義されている。

5.6.1 MqStatus 構造体

本構造体は、MqStat 列挙型定数を利用しており、ヘッダアクセスの成功／失敗を記録している。ヘッダへのアクセスが正常に終了した場合には、各ヘッダに対応する構造体メンバに MQ_SUCCESS が記録され、失敗した場合には MQ_FAIL が代入される。なお、ヘッダにアクセスしなかった場合には、該当する構造体メンバには MQ_NOACCESS が代入される。この代入は、プログラム初頭において MqInit() 関数 (C 言語)、MQINIT() サブルーチン (FORTRAN) が呼ばれた際に行なわれるので、この関数またはサブルーチンが呼ばれていない場合、本構造体の内容は保証されない。

MqStatus 構造体の内容は次の通りである。

```
struct {
    MqStat  AveAmp;       /* 平均振幅 */
    MqStat  Channels;     /* チャンネル数 */
    MqStat  DataCreDate;  /* データ作成日時 */
    MqStat  DataFormat;   /* 元ファイルのデータフォーマット */
    MqStat  DataModDate;  /* データ修正日時 */
    MqStat  DataType;     /* データ種別 */
    MqStat  EndRecord;    /* データの終了レコード */
    MqStat  FileType;     /* ファイルフォーマット */
    MqStat  MaxAmp;       /* 最大振幅 */
    MqStat  NumOfData;    /* データ数 */
    MqStat  ObsMode;      /* 観測モード */
    MqStat  OrgFileName;  /* 元データのファイル名 */
    MqStat  SampRate;     /* サンプリング間隔 */
    MqStat  Station;      /* 観測点 */
    MqStat  StartRecord;  /* データの開始レコード */
    MqStat  StartTime;    /* データの開始時刻 */
    MqStat  TapeNumber;   /* テープ番号 */
    } MqStatus;
```

5.6.2 MqLibVersion 構造体

本構造体には、MQLIB のバージョンが保存されている。本変数は、MqInit() 関数 (C 言語)、または MQINIT() サブルーチン (FORTRAN) によって初期化される。また MQLIB を利用して MQDB フォーマットファイルを作成する場合、本構造体の情報を利用して先頭行を作成する。

MqLibVersion 構造体は MqLibVer 型の構造体で (第?? 節参照)、そのメンバは次のようになっている。

メンバの型	メンバ名	内容
char	LibType[2]	MQDB ライブラリ識別子
char	Release	MQDB リリース番号
char	Major	MQDB メジャー番号
char	Minor	MQDB マイナー番号

5.6.3 MqFilDesc 構造体配列

本構造体配列は、MqOpen() または MQOPEN() 関数／サブルーチンによってオープンされたファイルについての情報を管理するための変数であり、配列は 16 個確保される。従って、MqOpen() または MQOPEN() を利用して同時にオープンできる MQDB フォーマットファイルの数は 16 個となる[3]。

MqFilDesc は MqFile 型構造体 (第 5.3.3 節参照) の配列で、配列の添字がファイル記述子の番号に対応する。

MqOpen() または MQOPEN() サブルーチンが呼ばれると、未使用で最も若いファイル記述子番号が新たに割り当てられ、必要な情報がセットされた上で、そのファイル記述子番号がプログラムに返される。この

[3] MqRead() 及び MqCreate() 関数を利用した場合には、このような制限はない

情報は、MqClose() または MQCLOS() 関数／サブルーチンによってファイル記述子の割当が解除され、番号が破棄されるまで変更することはできない。

Chapter 6

関数・サブルーチン仕様

本章では、MQLIB において定義されているアクセス用関数の仕様について説明する。本章の説明は、次のようになっている。

仕様 関数の型、及び引数が書かれている、スペースの関係で、ほとんどの関数について引数は型名だけを記してある。

引数 引数の型名及びその内容が書かれている。引数は左から順に説明される。例えば、`func( char *str1, char *str2 )` のような関数の場合、
`char *` は `str1` の説明であり、
`char *` は `str2` の説明となる。

また、FORTRAN インタフェースについては、C 言語用の上位インタフェースに合わせて記述されている。

6.1 上位関数インタフェース仕様

6.1.1 ファイル記述子の解放

C **未実装**

FORTRAN `MQCLOS( INTEGER*2 FD )`

C	FORTRAN	引数
char *	FD	ファイル記述子
返値	DATATYPE	データ種別

引数で指定されたファイル記述子の割当を解除する。ヘッダの読み書きに一時ファイルを使用していた場合には、そのファイルは削除される。該当するファイル記述子は初期化され、`MqOpen()` または `MQOPEN()` により再割当が可能となる。

ファイルを作成する場合、本関数／サブルーチンの呼び出しによって、ヘッダ用一時ファイルから MQDB フォーマットファイル用のヘッダが作られ、さらにデータが付加されて出力される。

6.1.2 MQDB フォーマットファイルの作成

仕様 `void MqCreate( char *, MqAccess *, MqHeader *, double * )`

引数

char *	出力ファイル名
MqAccess *	アクセス制御変数
MqHeader *	ヘッダ情報を格納する構造体
double *	実際のデータを格納する構造体

MqAccess 構造体及び MqHeader 構造体変数の情報に従って、ヘッダ文字列を作成し、double 配列に格納されたデータ出力する。出力ファイル名が NULL であった場合には、標準出力にデータが出力される。

MqAccess 構造体は、ヘッダのアクセスを制御する。本構造体変数に MQ_NO がセットされていれば、該当するヘッダは作成されない。但しどのような場合においても、先頭行及び「ファイルの作成日時」ヘッダだけは必ず作成される。また、ヘッダ作成中にエラーが起こった場合、そのエラー以降はヘッダの作成は行なわれない。なお、ヘッダの作成は「終了レコード」ヘッダを除き、ヘッダ文字列のアルファベット順に行なわれる。

ヘッダ作成が終了すると、本関数は MqCreData() 関数を呼び出し、データ部の作成が開始される。データ部は、MqHeader 構造体内の FileType メンバにセットされた値に従ったフォーマットで作成される。

6.1.3 MQLIB の初期化

仕様

C	unsigned short MqInit(char *)
FORTRAN	MQINIT(CHARACTER PROGSIGN)

引数

char *	プログラム署名
PROGSIGN	プログラム署名

返値 実際のデータを格納する double 型配列へのポインタ

本関数及びサブルーチンは、MQLIB の初期化を行なう。すなわち、本関数／サブルーチンが呼び出されることにより、状況変数 (第 5.6 節参照) が初期化される。本関数及びサブルーチンは、MQLIB を利用する際にはプログラム初頭で必ず呼び出しておく必要がある。

本関数及びサブルーチンを呼び出すと、状況変数は次のように初期化される。

- MqErr は、warn メンバが EMQ_WARNING にセットされる。すなわち、デフォルトの状態では、WARNING レベル以上のエラーの場合にエラーメッセージが出力されるようになっている。
- MqStatus は、全メンバが MQ_NOACCESS にセットされる。
- MqLibVersion は、全メンバが MQLIB の該当バージョン (ヘッダファイル mqdb.h に記述されているライブラリ・バージョン) になるように初期化される。
- MqFilDesc 構造体配列は、全メンバが次のように設定される。

InUse	MQ_NO (未使用の状態になる)
accmethod	MQ_TMP (一時ファイルを用いてヘッダへのアクセスを行なう)
iostream	MQ_FILE (通常のファイルへのアクセス)
ReadWrite	MQ_UNKNOWN (アクセス方法は未定)
filename	NULL

また、ライブラリ・バージョンは、上記と同じである。

MQDB フォーマットファイルを作成する場合には、引数として渡されたプログラム署名が用いられる。

6.1.4 ファイル記述子の割り当て

C 未実装

FORTRAN `MQOPEN( CHARACTER*2 FILENAME, INTEGER*2 RW, INTEGER*2 IO, INTEGER*2 FD )`

C	FORTRAN	引数
	FILENAME	ファイル名
	RW	ファイルの読み書きのモード
	IO	オープンするファイル
返値	FD	ファイル記述子

MQDB フォーマットファイルの入出力にファイル記述子を割り当て、アクセス用関数が使用できる状態にする。本関数／サブルーチンが呼び出されると、ヘッダの内容が一時ファイルに蓄えられ[1]、引数に指定されたファイルのオープンのモードでファイルがファイル記述子と結びつけられる。ファイルアクセスが正常に終了すると、ファイル記述子が返される。MQDB フォーマットファイルへのアクセスは、MQOPEN() 呼出後は全てファイル記述子によって行なわれる。

入出力に標準入出力を使用する場合には、ファイル名は NULL や空白文字などで構わない (無視される)。RW 及び IO は、それぞれ次のように変換される。

C	FORTRAN	ファイルの読み書きのモード
MQ_READ	1	(読み出し)
MQ_WRITE	2	(書き込み)

C	FORTRAN	オープンするファイル
MQ_FILE	1	通常のファイル
MQ_STDIN	2	標準入力
MQ_STDOUT	3	標準出力
MQ_STDOUT	4	標準エラー出力

6.1.5 MQDB フォーマットファイルの読み込み

仕様 `double MqRead( char *, MqAccess *, MqHeader * )`

引数	`char *`	出力ファイル名
	`MqAccess *`	アクセス制御変数
	`MqHeader *`	ヘッダ情報を格納する構造体

返値 実際のデータを格納する double 型配列へのポインタ

char * 変数で指定された月震データファイルを読み、ヘッダ情報を MqHeader 構造体変数に、データを double 変数の配列に格納して返値として返す。MqAccess 構造体は、ヘッダへのアクセスを制御する。本構造体に MQ_NO がセットされているヘッダについては読みとられない。char * に NULL が指定されていれば、標準入力からデータが読みとられる。

本関数はまず、ファイルの最初の 2 文字を読み、MQDB フォーマットファイルかどうかを判断する。但し、その判定によってデータベース用ファイルでないとされた場合でも、プログラムは終了しない。その後、本関数は先頭行、及びヘッダ行の読みとりを行い、ヘッダ行から得られたデータを MqHeader 構造体変数に格納する。

[1] 共有メモリなどを利用する機能は、今後のライブラリの改訂によって追加される予定である。

ヘッダの読み込みが終了すると、データの読みとりを行ない、得られたデータを double *変数に格納する。この格納はチャンネルの順番に行なわれる。すなわち、チャンネル 1 のデータは配列の $0 \sim n-1$ 番めまでに格納される。チャンネル 2 のデータは配列の $n \sim 2n-1$ 番めまでに格納される。MqHeader の NumOfData メンバに適切と思われるデータ個数が設定されている場合には、データ領域は自動的に拡張される。もしメモリの確保に失敗するなどの理由でデータが格納できなかった場合には、NULL が返される。

6.1.6 「チャンネル数」ヘッダの読み込み

C 未実装

FORTRAN MQRC(INTEGER*2 FD, INTEGER*2 CHANNELS)

C	FORTRAN	引数
char *	FD	ファイル記述子
返値	CHANNELS	チャンネル数

引数で指定されたファイル記述子に対応する「チャンネル数」ヘッダを読み込み、チャンネル数を返す。

6.1.7 「データフォーマット」ヘッダの読み込み

C 未実装

FORTRAN MQRDF(INTEGER*2 FD, INTEGER*2 DATAFORM)

C	FORTRAN	引数
char *	FILENAME	ファイル記述子
返値	CHANNELS	データフォーマット

引数で指定されたファイル記述子に対応する「データフォーマット」ヘッダを読み込み、データフォーマットを表す値を返す。

MQRDF() において、返されるデータフォーマットの値と、実際のデータフォーマットとの対応は、次の表の通りである。なお、ヘッダに不正な値が格納されていた場合には、MQRDF() では DATAFORM に 0 が格納される。

C	FORTRAN	データフォーマット
MQ_OLD	1	‘‘OLD’’ フォーマット
MQ_NEW	2	‘‘NEW’’ フォーマット

6.1.8 「データ種別」ヘッダの読み込み

C 未実装

FORTRAN MQRDF(INTEGER*2 FD, INTEGER*2 DATATYPE)

C	FORTRAN	引数
char *	FD	ファイル記述子
返値	DATATYPE	データ種別

引数で指定されたファイル記述子に対応する「データフォーマット」ヘッダを読み込み、データフォーマットを返す。返されるデータ種別の値と、実際のデータ種別との対応は、次の表の通りである。なお、ヘッダに不正な値が格納されていた場合には、MQRDT() では DATATYPE に 0 が格納される。

C	FORTRAN	データ種別
MQ_LP	1	LP（長周期データ）
MQ_SP	2	SP（短周期データ）
MQ_TIDAL	3	TIDAL（潮汐データ）

6.1.9 「終了レコード番号」ヘッダの読み込み

C　　未実装

FORTRAN　MQRERC(INTEGER*2 FD, INTEGER*2 ENDREC)

C	FORTRAN	引数
char *	FILENAME	ファイル記述子
返値	ENDREC	終了レコード番号

引数で指定されたファイル記述子に対応する「終了レコード番号」ヘッダを読み込み、終了レコード番号の値を返す。

6.1.10 「ファイル種別」ヘッダの読み込み

C　　未実装

FORTRAN　MQRFT(INTEGER*2 FD, INTEGER*2 FILETYPE)

C	FORTRAN	引数
char *	FD	ファイル記述子
返値	FILETYPE	ファイル種別

引数で指定されたファイル記述子に対応する「ファイル種別」ヘッダを読み込み、ファイル種別を返す。

返されるファイル種別の値と、実際のファイル種別との対応は、次の表の通りである。なお、ヘッダに不正な値が格納されていた場合には、MQRFT() では FILETYPE に 0 が格納される。

C	FORTRAN	ファイル種別
MQ_FULLTEXT	1	フルテキスト形式
MQ_COMPOSITE	2	バイナリ混在形式
MQ_XDR	3	XDR 形式

6.1.11 「データ個数」ヘッダの読み込み

C　　未実装

FORTRAN　MQRNOD(INTEGER*2 FD, INTEGER*4 NOD)

C	FORTRAN	引数
char *	FD	ファイル記述子
返値	NOD	データ個数

引数で指定されたファイル記述子に対応する「データ個数」ヘッダを読み込み、データ個数を返す。

6.1.12 「観測モード」ヘッダの読み込み

C 未実装

FORTRAN `MQROM( INTEGER*2 FD, INTEGER*2 OBSMODE )`

C	FORTRAN	引数
char *	FD	ファイル記述子
返値	CHANNELS	観測モード

引数で指定されたファイル記述子に対応する「観測モード」ヘッダを読み込み、観測モードを表す値を返す。

返される観測モードの値と、実際の観測モードの対応は、次の表の通りである。なお、ヘッダに不正な値が格納されていた場合には、`MQROM()` では `OBSMODE` に `0` が格納される。

C	FORTRAN	観測モード
MQ_PEAKED	1	PEAKED MODE
MQ_FLAT	2	FLAT MODE

6.1.13 「観測点」ヘッダの読み込み

C 未実装

FORTRAN `MQRSTN( INTEGER*2 FD, INTEGER*2 STATION )`

C	FORTRAN	引数
char *	FD	ファイル記述子
返値	STATION	観測点番号

引数で指定されたファイル記述子に対応する「観測点」ヘッダを読み込み、観測点番号の値を返す。

6.1.14 「開始レコード番号」ヘッダの読み込み

C 未実装

FORTRAN `MQRSRC( INTEGER*2 FD, INTEGER*2 STARTREC )`

C	FORTRAN	引数
char *	FD	ファイル記述子
返値	STARTREC	開始レコード番号

引数で指定されたファイル記述子に対応する「開始レコード番号」ヘッダを読み込み、開始レコード番号の値を返す。

6.1.15 「データ開始時刻」ヘッダの読み込み

C 未実装

FORTRAN
```
MQRST( INTEGER*2 FD, INTEGER*2 YEAR, INTEGER*2, DAY, INTEGER*2 HOUR,
INTEGER*2 MINUTE, INTEGER*2 SECOND, INTEGER*2 MS )
```

C	FORTRAN	引数
char *	FD	ファイル記述子
返値		開始レコード番号を格納した MqTime 構造体
	YEAR	年 (西暦)
	DAY	日 (365 日単位)
	HOUR	時 (24 時間単位)
	MINUTE	分
	SECOND	秒
	MS	ミリ秒

引数で指定されたファイル記述子に対応する「データ開始時刻」ヘッダを読み込み、データ開始時刻の年、日、字、分、秒、ミリ秒の値を返す。

6.1.16 「テープ番号」ヘッダの読み込み

[注意]C 言語用インタフェースは未実装

C 未実装

FORTRAN MQRTN(INTEGER*2 FD, INTEGER*4 TAPE)

C	FORTRAN	引数
char *	FD	ファイル記述子
返値	TAPE	テープ番号

引数で指定されたファイル記述子に対応する「テープ番号」ヘッダを読み込み、テープ番号の値の値を返す。

6.2 中位関数インタフェース仕様

6.2.1 「平均振幅」ヘッダの読み込み

仕様 MqAmp MqGetAveAmp(char *)

引数 char * ヘッダ文字列

返値 平均振幅

ヘッダ文字列から「平均振幅」ヘッダを探し、データ文字列中のコンマ数を元にチャンネル数を判断し、必要なデータを切り出した上で MqAmp 変数に格納して返す。

6.2.2 「チャンネル数」ヘッダの読み込み

仕様 unsigned short MqGetChannels(char *)

引数 char * ヘッダ文字列

返値 チャンネル数

ヘッダ文字列から「チャンネル数」ヘッダを探し、情報文字列を切り出してチャンネル数に相当する数字を返す。

6.2.3 「データフォーマット」ヘッダの読み込み

仕様 `MqFormat MqGetDataFormat( char * )`

引数 `char *` ヘッダ文字列

返値 データフォーマットを表す変数。

「データフォーマット」ヘッダを探し、該当するデータフォーマットを読み出してそれを `MqFormat` 変数として返す。

6.2.4 「データ種別」ヘッダの読み込み

仕様 `MqType MqGetDataType( char * )`

引数 `char *` ヘッダ文字列

返値 データ種別を表す変数

「データ種別」ヘッダを探し、該当するデータ種別を読みだし、それを `MqType` 変数として返す。

6.2.5 「終了レコード番号」ヘッダの読み込み

仕様 `unsigned long MqGetEndRecord( char * )`

引数 `char *` ヘッダ文字列

返値 終了レコードを表す数字

「終了レコード番号」ヘッダを探し、情報文字列を数字に変換して返す。

6.2.6 ヘッダ文字列の切り出し

仕様 `char * MqGetHeader( FILE * )`

引数 `char *` 入力ファイル記述子

返値 処理後のヘッダ文字列

入力ファイルからヘッダ部分を読みだし、コメント、空白部分などを削除し、ヘッダだけになった文字列を返す。正常にヘッダ文字列を切り出せなかった場合には `NULL` が返される。

6.2.7 「最大振幅」ヘッダの読み込み

仕様 `MqAmp MqGetMaxAmp( char * )`

引数 `char *` ヘッダ文字列

返値 最大振幅

ヘッダ文字列から「最大振幅」ヘッダを探し、データ文字列中のコンマ数を元にチャンネル数を判断し、必要なデータを切り出した上で `MqAmp` 変数に格納して返す。

6.2.8 「データ個数」ヘッダの読み込み

仕様 `unsigned long MqGetNumOfData( char * )`

引数 `char *` ヘッダ文字列

返値 データ個数を表す数字

「データ個数」ヘッダを探し、情報文字列を数字に変換して返す。

6.2.9 「観測モード」ヘッダの読み込み

仕様 `MqMode MqGetObsMode( char * )`

引数 `char *` ヘッダ文字列

返値 観測モードを表す変数

「観測モード」ヘッダを読み込み、情報文字列から観測モードを読みだし、`MqMode` 変数の形で返す。

6.2.10 「元のデータファイル名」ヘッダの読み込み

仕様 `void MqGetOrgFile( char *, char * )`

引数
`char *` ヘッダ文字列
`char *` 元のファイル名

「元のデータファイル名」ヘッダを読み込み、情報文字列を元のデータファイル名として返す。

6.2.11 「サンプリング間隔」ヘッダの読み込み

仕様 `MqAmp MqGetSampRate( char * )`

引数 `char *` ヘッダ文字列

返値 サンプリング間隔

ヘッダ文字列から「サンプリング間隔」ヘッダを探し、データ文字列中のコンマ数を元にチャンネル数を判断し、必要なデータを切り出した上で `MqAmp` 変数に格納して返す。

6.2.12 「開始レコード番号」ヘッダの読み込み

仕様 `unsigned long MqGetStartRecord( char * )`

引数 `char *` ヘッダ文字列

返値 開始レコード番号を表す変数

「開始レコード番号」ヘッダを探し、情報文字列を数字に変換して返す。

6.2.13 「データのスタート時刻」ヘッダの読み込み

仕様 `MqTime MqGetStartTime( char * )`

引数 `char *` ヘッダ文字列

返値 データのスタートする時刻

「データのスタート時刻」ヘッダを探し、情報文字列を数字に変換して返す。

6.2.14 「観測ステーション」ヘッダの読み込み

仕様 `unsigned short MqReaStation( char * )`

引数 `char *` ヘッダ文字列

返値 観測ステーションを表す数字

「観測ステーション」ヘッダを探し、情報文字列を数字に変換して返す。

6.2.15 「テープ番号」ヘッダの読み込み

仕様 `unsigned short MqGetTapeNumber( char * )`

引数 `char *` ヘッダ文字列

返値 テープ番号を表す変数

「テープ番号」ヘッダを探し、情報文字列を数字に変換して返す。

6.2.16 先頭行の読み込み

仕様 `MqLibInfo MqGetTopLine( FILE * )`

引数 `FILE *` 入力ファイルのファイル記述子

返値 ライブラリ情報構造体

指定された入力ファイルから先頭行を読み、指定されたファイルが MQDB フォーマットであるかどうかを判断する。もし MQDB フォーマットであると判断された場合には、ライブラリ番号、プログラム署名を読み出す。この判断は、先頭２文字が `@@` であるかどうかで行なう。もし MQDB フォーマットでない場合には、バージョン番号は全て `0` となり、プログラム署名の変数には `NULL` が返される。

6.2.17 「平均振幅」ヘッダ文字列の作成

仕様 `char * MqPutAveAmp( MqAmp )`

引数 `MqAmp` 平均振幅

返値 「平均振幅」ヘッダ文字列へのポインタ

「平均振幅」ヘッダ用の文字列を作成する。

6.2.18 「チャンネル数」ヘッダ文字列の作成

仕様 `char * MqCreChannels( unsigned short )`

引数 `unsigend short` 観測モード

返値 「チャンネル数」ヘッダ文字列へのポインタ

「チャンネル数」ヘッダ文字列を作成する。

6.2.19 「データ作成日時」ヘッダ文字列の作成

仕様 `void MqPutCreateDate( void )`

引数 `char *` ヘッダ文字列

「データ作成日時」ヘッダ用の文字列を作成する。本関数は、`MqCreate()` 内では自動的に呼び出される。

6.2.20 データ部の作成

仕様 `void MqPutData( FILE *, double *, MqHeader * )`

引数
- `FILE *` ファイル構造体
- `double *` データ
- `MqHeader *` ヘッダ構造体

データ部を作成する。`MqHeader` 構造体変数に格納されているチャンネル数とデータ数の情報を用いて、データの行数を決定した後、チャンネル数に応じてデータを出力する。`FILE` 構造体が `NULL` であれば、出力は標準出力に対して行なわれる。それ以外の場合には、`FILE` 構造体変数で指定されるファイルに対して出力される。

6.2.21 「データフォーマット」ヘッダ文字列の作成

仕様 `char * MqPutDataFormat( MqFormat )`

引数 `char *` ヘッダ文字列

返値 「データフォーマット」ヘッダ文字列へのポインタ

`MqFormat` 構造体変数の内容に従い、「データフォーマット」ヘッダ用の文字列を作成する。

6.2.22 「データ種別」ヘッダの作成

仕様 `char * MqPutDataType( MqType )`

引数 `MqType` 月震のデータ種別

返値 「データ種別」ヘッダ文字列へのポインタ

`MqType` 構造体変数の内容に従って、「データ種別」ヘッダ用の文字列を作成する。

6.2.23 「終了レコード番号」ヘッダ文字列の作成

仕様 `char * MqPutEndRecord( unsigned long )`

引数 `unsigned long` 終了レコード番号

返値 「終了レコード番号」ヘッダ文字列へのポインタ

引数として受けとったレコード番号から、「終了レコード番号」ヘッダ用の文字列を作成する。

6.2.24 「最大振幅」ヘッダ文字列の作成

仕様 `char * MqPutMaxAmp( MqAmp )`

引数 `MqAmp` 最大振幅

返値 「最大振幅」ヘッダ文字列へのポインタ

「最大振幅」ヘッダ用の文字列を作成する。`MqAmp` 変数の内容を文字列化して、ヘッダ文字列に付加する。

6.2.25 「データ個数」ヘッダ文字列の作成

仕様 `char * MqPutNumOfData( unsigned long )`

引数

`char *`	ヘッダ文字列
`unsigned long`	データ個数

返値 「データ個数」ヘッダ文字列へのポインタ

「データ個数」ヘッダ用の文字列を作成する。

6.2.26 「観測モード」ヘッダ文字列の作成

仕様 `char * MqPutObsMode( MqMode )`

引数 `MqMode` 観測モード

返値 「観測モード」ヘッダ文字列へのポインタ

「観測モード」ヘッダ用の文字列を作成する。

6.2.27 「元のデータファイル名」ヘッダ文字列の作成

仕様 `char * MqPutOrgFile( char * )`

引数 `char *` 元のデータファイル名

返値 「元のデータファイル名」ヘッダ文字列へのポインタ

「元のデータファイル名」ヘッダ用文字列を作成する。

6.2.28 「サンプリング間隔」ヘッダ文字列の作成

仕様 `char * MqPutSampRate( MqAmp )`

引数 `MqAmp` サンプリング間隔

返値 「サンプリング間隔」ヘッダ文字列へのポインタ

「サンプリング間隔」ヘッダ用の文字列を作成する。`MqAmp` 変数の内容を文字列化して、ヘッダ文字列に付加する。

6.2.29 「開始レコード番号」ヘッダの作成

仕様 `char * MqPutStartRecord( unsigned long )`

引数 `unsigned long` 開始レコード番号

返値 「開始レコード番号」ヘッダ文字列へのポインタ

引数として受けとったレコード番号から、「開始レコード番号」ヘッダ用の文字列を作成する。

6.2.30 「データのスタート時刻」ヘッダ文字列の作成

仕様 `char * MqPutStartTime( MqTime )`

引数 `MqTime` データのスタート時刻を格納した構造体

返値 「データのスタート時刻」ヘッダ文字列へのポインタ

「データのスタート時刻」ヘッダ用の文字列を作成する。

6.2.31 「観測ステーション」ヘッダ文字列の作成

仕様 `char * MqPutStation( unsigned short )`

引数 `unsigned short` 観測点の番号を格納した変数 (12 〜 16)

返値 「観測ステーション」ヘッダ文字列へのポインタ

「観測ステーション」ヘッダ用の文字列を作成する。

6.2.32 「テープ番号」ヘッダ文字列の作成

仕様 `void MqPutTapeNumber( unsigned short )`

引数 `unsigned short` テープの番号を格納した変数

返値 「テープ番号」ヘッダ文字列へのポインタ

「テープ番号」ヘッダ用文字列を作成する。

6.2.33 先頭行文字列の作成

仕様 `void MqPutTopLine( char *header, char *progsign )`

引数
`char *header` ヘッダ文字列
`char *progsign` プログラム署名

先頭行として利用できる文字列を作成し、`*header` にセットする。ライブラリのバージョンは、状況変数 `MqLibVersion` よりとられる。従って、本関数の実行前に、必ず `MqInit()` 関数を実行しておかなくてはならない。

6.3 下位関数インタフェース仕様

6.3.1 ヘッダ文字列の作成

仕様 `char * MqCatInfoData( char *, char * )`

引数
`char *` 情報文字列
`char *` データ文字列

返値 ヘッダ文字列

情報文字列とデータ文字列の間に分離記号 : を加え、ヘッダ文字列を作成して返値として返す。もし作成中にエラーが発生した場合には、`NULL` が返される。

6.3.2 エラー番号のクリア

仕様 `void MqClearError( void )`

エラー番号をクリアする。ほとんどの `MQLIB` 関数では、最初にこの関数が呼び出されている。なお、本関数はマクロとして実現されており、実際の形は

```
MqErr.errno = 255
```

である。

6.3.3 文字列から double 型配列への変換

仕様 `void MqConvCharDouble( char *, MqAmp * )`

引数
`char *` 文字列
`MqAmp *` 文字列から変換された `double` 型データを収めるための変数。

文字列中のセパレータ , の位置を調べ、それを元にしてセパレータで区切られた数値列を `double` 型の配列とみなし、変換して格納する。

6.3.4 セパレータ数のカウント

仕様 `unsigned short MqCountSeparator( unsigned short, char * )`

引数

`unsigned short`	分離記号を表す文字 `1` 文字。
`char *`	ヘッダ文字列

返値 捜し出された分離記号の数。

セパレータを文字列 `char *` 変数中から探し、その数を返す。

6.3.5 ヘッダ文字列からの１行の切り出し

仕様 `void MqCutLine( char *, char *, long )`

引数

`char *`	ヘッダ文字列
`char *`	切り出された１行
`long`	行の先頭の位置

`long` 変数に格納されたヘッダ行の先頭の情報に基づき、そこから改行を表す部分までを複写し、C 言語で表される文字列の形、すなわち最後に'\0'が付加された文字列の形として返す。`long` 変数はポインタではなくて、文字列の先頭から数えた該当するヘッダまでの文字数と考えれば良い。

6.3.6 コメントの先頭文字かどうかを判断

仕様 `void MqIsComStart( int )`

引数 `int` チェックしたい文字

返値 先頭文字であれば `MQ_TRUE`、そうでなければ `MQ_FALSE`

`int` で示される文字がコメントの先頭文字として有効かどうかを判断し、有効であれば `MQ_TRUE`、そうでなければ `MQ_FALSE` を返す。なお、`MQ_TRUE` 及び `MQ_FALSE` はいずれもマクロ定義された変数である。また、本関数はマクロによって実現されている。

6.3.7 行末記号の出力

仕様 `void MqPutLineEnd( char * )`

引数 `char *` ヘッダ文字列

引数であるヘッダ文字列の最後に行末記号 (`CR + NL`) を付加する。なお、この関数はマクロによって実現されており、実際の形は、

```
strcat( x, MqLineEnd )
```

である。

6.3.8 ヘッダ用一時ファイルの読み込み

仕様 `char * MqGetdTmpFile( unsigned short )`

引数 `unsigned short` ファイル記述子

返値 読み込まれたヘッダ用一時ファイルの内容を含む文字列

本関数は、`MqOpen()` あるいは `MQOPEN()` 関数／サブルーチンによって作成された、ヘッダ用一時ファイルを読み込む。読み込まれるファイルは、引数で指定されたファイル記述子に対応する一時ファイルである (第 ?? 節参照)。ファイルの読み込みに成功すると、その内容が `char *` 型変数として読み込まれ、返値として返される。もし読み込み中にエラーが発生した場合には、`NULL` が返される。

6.3.9 情報文字列の検索

仕様 `long MqSearchHeader( char *, char * )`

引数
`char *` ヘッダ文字列
`char *` 情報文字列

ヘッダ文字列内で情報文字列を検索し、その位置を `long` 変数として返す。もし、文字列が見つからなかったり、エラーが発生した場合には、`-1` が返される。

6.3.10 情報文字列とデータ文字列の分離

仕様 `void MqSeparateInfoData( char *, char * )`

引数
`char *` ヘッダ文字列
`char *` データ文字列

情報文字列とデータ文字列の分離記号:を頼りにデータ文字列をヘッダ文字列全体から分離する。

6.3.11 文字列から MqTime 構造体変数への変換

仕様 `MqTime MqSeparateMqTime( char * )`

引数 `char *` データ文字列

本関数は、数字が次のような形で並んでいる文字列を、`MqTime` 構造体変数へ変換する。各文字列は、`65536` 以下の自然数でなければならない。各文字列が表現する数値の許容範囲は、`MqTime` 構造体による許容範囲と同じである。

```
year day hour minute second ms
(例)
1972 223 22 10 43 34
```

6.3.12 エラー番号のセット

仕様 `void MqSetError( unsigned long )`

引数 `unsigned long` エラー番号

エラー番号をセットする。この関数はマクロとして実現されており、実際の形は

```
MqErr.errno = x
```

である。`x` は引数である。

6.3.13 ファイル記述子のセット

仕様 `short MqSetFilDesc( char *, MQRW, MQIO );`

引数
- `char *` ファイル名
- `MQRW` ファイルの読み書きのモード
- `MQIO` オープンするファイル

返値 ファイル記述子

ファイル記述子を、引数に与えられたモード値でセットする。本関数が呼び出されると、まず未使用の最も若い番号のファイル記述子が探され、もし見つかれば（見つからなければエラーとなる)、そのファイル記述子に、引数で与えられた読み書きのモード、ファイル入出力の種類がセットされ、全て正しくセットされれば、ファイル記述子を表す数字が返値として返される。もし途中でエラーが発生した場合には、`-1` が返される。

6.3.14 関数番号のセット

仕様 `void MqSetFunc( unsigned long )`

引数 `unsigned long` 関数番号

関数番号をセットする。この関数はマクロとして実現されており、実際の形は

```
MqErr.func = x
```

である。`x` は引数である。

6.4 ユーティリティ関数

6.4.1 平均振幅の計算

仕様 `double MqCalcAveAmp( double *, unsigned long start, unsigned long num )`

引数
- `double *` 実際のデータ
- `unsigned long` 計算を開始するデータ点
- `unsigned long` 計算するデータ数

返値 平均振幅

`double *`型配列で示されるデータの `start` で示される箇所から、`num` 個のデータについて平均値を計算する。

6.4.2 最大振幅の計算

仕様 `double MqCalcMaxAmp( double *, unsigned long start, unsigned long num )`

	`double *`	実際のデータ
引数	`unsigned long`	計算を開始するデータ点
	`unsigned long`	計算するデータ数

返値 最大振幅

`double *`型配列で示されるデータの `start` で示される箇所から、`num` 個のデータについて最大振幅を探し出す。

6.4.3 データベース用ファイル名の作成

仕様 `char * MqCreFileName( unsigned short, MqTime, MQDT )`

	`unsigned short`	観測点番号
引数	`MqTime`	データ開始時刻
	`MQDT`	月震データの種別

返値 作成されたファイル名

引数として渡された観測点番号、データ開始時刻、データ種別を元にして、月震データベース用のファイル名を作成する。各引数の内容は、`MqRead()`、あるいは `MqReadStation()`、`MqReadStartTime()`、`MqReadDataType()` などから得ることができる。引数が不正であった場合、例えば、観測点番号が `11` 以下 `17` 以上であったり、`MqTime` 構造体の中に時刻として不適当な値が代入されていた場合などには、ファイル名は作成されず、`NULL` が返される。それ以外の場合には、月震データベース既約に基づいたファイル名が返される。但し、ファイル名の第 `8` 文字については、必ず `0` となっているので、データベースの規約に合わせるためには、`MQLIB` のファイルをインストールするプログラムで、適切な名前に変換する必要がある。

6.4.4 MqFilDesc 変数の整合性チェック

仕様 `void MqCheckFilDesc( unsigned short, MQRW )`

	`unsigned short`	チェックを行なうファイル記述子
引数	`MQRW`	ファイルがオープンされているモード (読み出しか書き出しか)

引数として渡されたファイル記述子の内容について、正しい内容が書き込まれているかどうかチェックを行なう。渡されたファイル記述子が `MQFILDESCMAX` 以上の値であったり、使用されていないファイル記述子であった場合にはエラーとなる。また、該当するファイル記述子にセットされているファイルのオープンのモード (`MQRW` 変数) と、引数として渡された `MQRW` 変数のオープンのモードが異なる場合にもエラーとなる。もちろん、`MQRW` として不正な値がセットされている場合にもエラーとなる。

6.4.5 MqLibVer 構造体の内容の判定

仕様 `void MqCheckLibVer( MqLibVer )`

引数 `MqLibVer` ライブラリ情報を格納した変数

`MqLibVer` 構造体に格納した変数が `MQDB` の規約に合致しているかどうかを判定する。もし合致していなければ、`MqErr` 構造体に違反内容に会わせたエラー番号が格納される。規約に合っていれば、`MqErr.errno` には `EMQNOERROR` が格納される。

6.4.6 観測モードの判定

仕様 `MqMode MqCheckObsMode( MqTime, unsigned short )`

引数

`MqTime`	データのスタート時刻
`unsigned short`	観測点を表す数値

返値 観測モード

`MqTime` 変数に格納されているデータのスタート時刻と観測点の番号から、そのデータが `peaked` モードで観測されていたか `flat` モードで観測されていたかを判別し、その値を返す。

6.4.7 サンプリング間隔の生成

仕様 `double MqCheckSampRate( MQDT )`

引数 `MQDT` 月震のデータ種類

返値 サンプリング間隔

月震のデータ種別に応じて、サンプリング間隔の値を返す。

6.4.8 観測点番号のチェック

仕様 `void MqCheckStation( unsigned short )`

引数 `MqTime *` チェックされる変数

`unsigned short` 変数の内容が、アポロの観測点番号として妥当な値かどうか、すなわち、`12` 〜 `16` の範囲内にあるかどうかを調べる。もしその範囲内にない場合には、それに応じたエラー値がグローバル変数 `MqErr` にセットされる。

6.4.9 MqTime 変数の内容のチェック

仕様 `void MqCheckTime( MqTime * )`

引数 `MqTime *` チェックされる変数

`MqTime` 変数の内容をチェックし、妥当な値が各変数に代入されているかどうかをチェックする。もし代入されていない場合には、それに応じたエラー値がグローバル変数 `MqErr` にセットされる。

6.4.10 2 つの MqTime 変数の時間差の出力

仕様 `double MqDiffTime( MqTime time2, Mqtime time1 )`

引数

`MqTime *time2`	引かれる時間
`MqTime *time1`	引く時間

返値 マイクロ秒単位で計算した時間差

2 つの `MqTime` 変数の時間差、`time2 - time1` を計算し、マイクロ秒単位の時間差を返す。

6.4.11 エラーメッセージの出力

仕様 void MqPerror(char *)

引数 char * エラー文字列

エラー構造体 MqError にセットされたエラー内容を解釈し、エラーメッセージを出力する。また、もしエラーレベルが EMQ_FATAL であった場合には、後述するエラーメッセージ出力後に、プログラムを終了する。

本関数により出力されるエラーメッセージは、次のような形となる。

MQLIB error (level LEVEL) at func FUNC:
エラーメッセージ

ここで、LEVEL はエラーのレベルを示し、FUNC はエラーが発生した関数を表す。

エラーメッセージは、もし引数*errstr が NULL でなければ、次のような形となる。

[*errstr で表されたメッセージ]: [必要なエラーメッセージ]

もし *errstr が NULL であれば、エラーメッセージは次のようになる。

[必要なメッセージ]

6.4.12 MqAccess 変数を全て MQ_YES にセット

仕様 void MqSetAccessYes(MqAccess *)

引数 MqAccess * アクセス制御変数

引数の MqAccess 変数を全て MQ_YES にセットする。本関数は、全てのヘッダを読み出したり、全てのヘッダを作成する場合に利用できる。

6.4.13 MqAccess 変数を全て MQ_NO にセット

仕様 void MqSetAccessNo(MqAccess *)

引数 MqAccess * アクセス制御変数

引数の MqAccess 変数を全て MQ_NO にセットする。本関数を利用することにより、特定のヘッダだけを作成したり、読み出したりすることができる。すなわち、この関数を呼んだ後に、必要なヘッダにだけ MQ_YES をセットすることにより特定ヘッダへのアクセスを実現させるために用いることが可能である。

6.4.14 エラー出力レベルの制御

仕様 void MqSetWarnLevel(unsigned short)

引数 unsigned short エラー出力レベル

引数のエラー出力レベル以上のエラーでのみエラーメッセージが出力されるように、エラー構造体を変更する。なお、引数はなるべくマクロで与えるようにすべきである。以下の引数を与えることが可能である。これ以外の数値を与えた場合にはエラーとなり、デフォルト値 (EMQ_WARNING) がセットされる。

引数	動作
EMQ_NOTICE	NOTICE レベルを含む、全エラーのメッセージを出力する。
EMQ_WARNING	WARNING 以上のレベルでエラーメッセージを出力する。
EMQ_FATAL	FATAL レベルのエラーの場合のみエラーメッセージを出力する。
EMQ_NOMESSAGE	いかなるエラーが発生してもエラーメッセージを出力しない。

Chapter 7

ユーティリティ・プログラム

MQLIB の性能テスト、及び簡単な月震データの処理を行なうために、いくつかのユーティリティプログラムが配布パッケージに添付されている。ここではこれらの利用方法について説明する。

なお、これらのプログラムのインストール方法については、付録の「ライブラリのインストール」に述べられている。

7.1 makedb(月震データベース構築コマンド)

makedb は、アポロ月震データファイルから MQLIB 用のデータファイルを生成するためのシェルスクリプトである。このシェルスクリプトからは、次の２つのプログラムが起動される。

dbconv アポロ月震データファイルを読み込み、MQLIB 用データファイルに変換する。また、データファイル内のデータ開始時刻等を一覧表示することもできる。

reformdb 月震データファイルを整形し、ヘッダを付加する。

dbconv、reformdb 共に、単独で実行することも可能である。

7.1.1 dbconv(アポロ月震データファイル読み込みユーティリティ)

dbconv は、アポロ月震データファイルを読み込み、必要な情報をヘッダ化し、MQLIB 用データファイルを生成する。

コマンドの形式は次の通りである。

```
dbconv [ -tlsnvV ] filename [ startrec ] [ endrec ]
```

ここで、startrec はデータ読み込みを開始するレコード番号、endrec はデータ読み込みを終了するレコード番号で、指定がなければファイル全体を読み込む。なお、startrec と endrec を指定する際には、必ず両方を指定する必要がある。片方だけを指定するとエラーとなる。

オプションについては次の通りである。これらは複数指定することが可能である。

t 潮汐データのみを出力する。

l 長周期データのみを出力する。

s 短周期データのみを出力する。

n いずれのデータも出力しない。これは、後述する v オプションと組み合わせ、月震のヘッダデータ一覧表を出力するために利用する。

v 冗長 (verbose) モード。このオプションが指定されると、指定されたパラメータ、及び、読み込んだヘッダの情報を一覧表の形で出力する。出力されるヘッダの情報は、レコード番号、PSE ナンバー、オリジナルのテープ番号、レコードの先頭の日付、フォーマット、前のレコードとの時間差である。前のレコードとの時間差が 1.0 秒以上ある場合には、dbconv はレコードの区切りとみなし、セパレータを出力する。

V 超冗長 (very verbose) モード。冗長モードに加え、生成された一時ファイル名なども出力する。これはデバッグ用であり、通常は v を指定する。

7.1.2 reformdb(月震データベースファイル整形ユーティリティ)

reformdb は、月震データベースファイルにヘッダを付加するためのプログラムで ある。

7.2 mqsac(SAC フォーマット変換ユーティリティ)

mqsac は、MQDB フォーマットのデータファイルを、汎用のシグナル解析ユーティリティ、SAC (Seismic Analyze Code) で読みだし可能なフォーマットに変換するプログラムである。実行形式は次の通りである。

```
mqsac mqdb-file sac-file
```

ここで、mqdb-file は MQDB フォーマットのファイル名、sac-file は SAC フォーマットのファイル名である。なお、現在のところ変換の方式は MQDB ファイルから SAC フォーマットのファイルへの変換のみで、逆はできない。

7.3 wmas(画面表示ユーティリティ)

wmas (WaveMaster) は、月震データベースファイルを X Window System サーバが稼働しているマシンの端末画面上に表示するコマンドである。

7.4 wpr(プリンタ出力ユーティリティ)

wpr は、月震データベースファイルをプリンタに出力するためのコマンドである。このプログラムは月震データベースファイルを読み込み、これをプリンタ制御言語に変換して、標準出力に出力する。変換できるプリンタ言語は、現在のところ LIPS II+のみである。出力の際には、振幅、時間のスケールを自由に指定することが可能である。

プリンタ制御言語は標準出力に出力されるので、プリンタに直接出力する場合には、以下のようなコマンドを入力しなければならない。

```
wpr 69233121.lp | lp -d lp
```

これは、69233121.lp という月震データベースファイルを lp と名付けられたプリンタに出力するための命令である。

このコマンドのオプションは次の通りである。

-v, -verbose 冗長モード。プログラムは標準エラー出力に、簡単なメッセージを出力する。

-V, -Verbose 超冗長モード。プログラムは標準エラー出力にデバッグ用のメッセージを出力する。ほとんどが診断情報なので、デバッグの場合以外はできれば指定しない方がよい。

-help ヘルプメッセージ出力。簡単なヘルプメッセージを出力して、プログラムは終了する。

-page N 出力ページ指定。N は出力したいページの番号である。このオプションを利用すると、特定のページ部分だけを出力することができる。

-a, -ampscale amp 振幅スケールを指定する。この振幅スケールは、1チャンネルの地震波形を描くウィンドウの半分の高さになる。従って、-ampscale 10.0 と指定した場合には、振幅スケールは 10.0(単位はファイル内で使用されている数値に従う) で描かれ、高さ 20.0 のウィンドウに規格化された波形が描かれることになる。

-t, -timescale time 時間スケールを指定する。time は1ページあたりの出力時間の長さを秒単位で指定した数値である。-timescale 3600.0 と指定した場合には、1ページあたり1時間になるように、データを適当にサンプリングして出力する。

7.5 wm2mq(WM1 フォーマットから MQDB フォーマットへの変換

本プログラムは、WaveMaster 1.xx 用の地震波形記述用フォーマットで記述されたデータファイルを、MQDB フォーマットのファイルに変換するプログラムである。

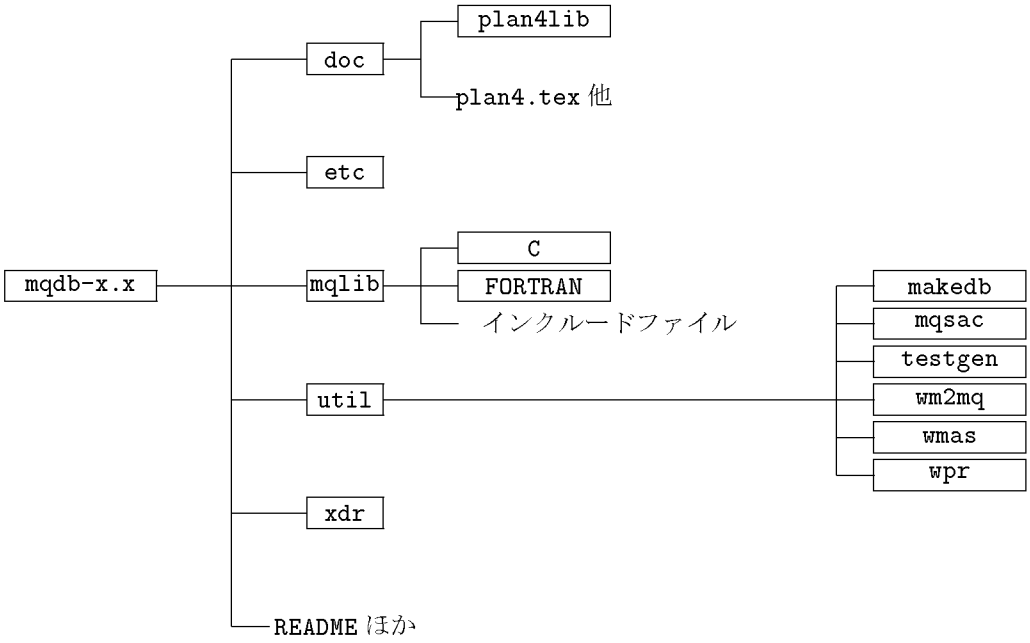
WaveMaster 1.xx においては、地震波データは2つのファイルに分かれて記述される。.wmas ファイルは、実際の地震波データが格納されているファイルであり、チャンネル毎に各成分がコンマで区切られた形で並んでいる。.info ファイルは地震波についての情報を記述したファイルであり、内容は任意である。本プログラムは、.info ファイル内の情報は利用していない。

Appendix A

配布パッケージのファイル構造について

第 B 章に述べる手順によって配布パッケージを展開すると、次のようなディレクトリ構造となっている。

Figure A.1: 配布パッケージのディレクトリ構造



Appendix B

ライブラリのインストール

本章では、ライブラリのインストール方法について述べる。ライブラリ及びユーティリティ構築用のファイルは、すべてC言語のソースファイルの形で配布されるので、利用するコンピュータにおいてこれらをコンパイルしなければならない。

ライブラリ及びユーティリティのインストールには、次のソフトウェアが必要となる。

1. C言語処理系。この処理系はANSI C対応である必要がある。
2. 配布されるアーカイブファイルを処理するためのユーティリティ・ソフトウェア。通常はtar及びgzip, gunzipである。tarコマンドはアーカイブ・ファイルを作成したり、アーカイブ・ファイルからファイルを抽出するコマンドである。gzip, gunzipコマンドはファイル圧縮／復元用のコマンドである。もし、システムにこれらのコマンドが付属していない場合には別途インストールする。特にgzip, gunzipについては、システムに付属していないため、必ず前もってインストールしておくこと。
3. makeコマンド。これはMakefileというファイルを利用して、ソフトウェアを構築するためのコマンドである。システムに付属していない場合には、別途インストールしておく必要がある。

また、以下のソフトウェアが存在すれば、フルインストールが可能となる。

1. FORTRAN言語処理系。MQLIBユーティリティの一部はFORTRAN言語で書かれているので、これらをコンパイルするために必要となる。これらのユーティリティを利用しない場合には必要ない。

以下、これらのソフトウェアが利用できるものと仮定して、インストールの手順について説明する。

1. 配布されているファイルを元のファイルの形に復元する。ソースファイルはtarコマンドによってアーカイブ化されており、gzipコマンドにより圧縮されて配布されている。このため、アーカイブファイルを復元した上で、そのアーカイブファイルからもとのファイルを展開する必要がある。

   この作業は、以下のようにして行う。ここでは、配布アーカイバ・ファイルの名前をmqdb-1.1.tar.gzとする。

   ```
   % gunzip mqdb-1.1.tar.gz
   % tar xvf mqdb-1.1.tar
   ```

   これらの作業が終了すると、mqdb-x.xというディレクトリが作成される。ここで、x.xはライブラリのバージョンであり、例えば、1.1となる。このディレクトリの下にソースファイルが解凍される。

2. mqdb-x.x ディレクトリに移り、次のコマンドを実行する。

3. mqlib ディレクトリ、及び util 下の各ディレクトリにある、Makefile の内容を確認し、必要なら、各システムの設定に合うように変更する。変更すべき箇所などは、各 Makefile 内の注釈を参照すること。

4. 変更が終了したら、mqdb-x.x ディレクトリに移り、次のコマンドを実行する。

```
% make
```

このコマンドにより、ライブラリ及びユーティリティ・コマンドのコンパイルが進行する。もし途中でエラーとなって失敗した場合には、Makefile で与えたオプションなどを再度確認する。

5. コンパイルが無事終了したら、続いて次のコマンドを実行する。

```
% make install
```

このコマンドにより、ライブラリ及びユーティリティがインストールされる。以上でライブラリ及びユーティリティのインストールは終了である。

Appendix C

エラー番号一覧

現在 MQLIB において定義されているエラー番号は以下の通りである。

エラーメッセージ	エラー番号
Too small year value in MqTime	EMQSMALLYEAR
Too large year value in MqTime	EMQLARGEYEAR
Too large day value in MqTime	EMQLARGEDAY
Too large hour value in MqTime	EMQLARGEHOUR
Too large minute value in MqTime	EMQLARGEMINUTE
Too large second value in MqTime	EMQLARGESECOND
Too large microsecond value in MqTime	EMQLARGEMS
Assigned file may not be a MQDB format	EMQILMQDBFORM
Data length of "Station" header is too long	EMQLONGSTATION
Error (unknown reason)	EMQERROR
Failed to open a file	EMQFILOPEN
File Format is not specified or unknown	EMQILFILEFORM
File length is only zero or one byte	EMQSHORTFILE
Header strings already exists	EMQEXISTHEADER
Header/Data separator not found	EMQNOSEPARATOR
Illegal character(s) found in "Station" header	EMQILSTATION
Illegal data format name	EMQILDATAFORM
Illegal data type	EMQILDATATYPE
Illegal error number	EMQILERRNUM
Illegal file type	EMQILFILETYPE
Illegal observation mode	EMQILOBSMODE
Illegal position	EMQILPOS
Illegal station number	EMQILSTNNUM
Information string is longer than header	EMQLONGINFO
Length of the data string is zero	EMQZERODATA
Length of the header line is zero	EMQZEROSTR
Length of the header string is zero	EMQZEROHDR
Length of the information string is zero	EMQZEROINFO
Length of the string ‘‘LibVersion’’ is zero	EMQZEROLIBV
Length of the string ‘‘LibVersion’’ too long	EMQLONGLIBV
Length of the string ‘‘ProgSign’’ is zero	EMQZEROPSGN
Length of the string ‘‘ProgSign’’ too long	EMQLONGPSGN
Memory allocation error	EMQMEMALLOC
Multiple header strings detected	EMQMULTINFO
No data separator , not need to call this function	EMQONEDATA
No separator character ’:’ found	EMQNOSEMICOLON
No separator character , found	EMQNOCOMMA
No such header found	EMQNOHEADER

また、次の表は、エラーメッセージからエラー番号を「逆引き」するためのものである。

エラーメッセージ	エラー番号
Non-numerical character(s) found in header	EMQNONUM
Null (maybe uninitialized) pointer detected	EMQNULLPTR
Number of channels in MqAmp are zero	EMQZEROAMPCHNL
Program exited with no error	EMQNOERROR
strcpy() error	EMQSTRCPY
strcat() error	EMQSTRCAT
The number of channel is zero	EMQZEROCHANNEL
The number of data is zero	EMQZERONOD
The number of data disagrees with channels	EMQILNOD

以下の表は、数字順に並べられたエラー番号である。

マクロ名	実際の番号	エラーレベル	エラー内容
EMQERROR	301		その他のエラー
EMQEXISTHEADER	258	WARNING	ヘッダ文字列が既に存在している
EMQFILOPEN	288	FATAL	ファイルのオープンに失敗した
EMQILDATAFORM	279	WARNING	不正なデータフォーマットである
EMQILDATATYPE	280	WARNING	不正なデータ種別である
EMQILERRNUM	257	WARNING	不正なエラー番号を使用している
EMQILFILEFORM	277	WARNING	ファイルフォーマットが確認できない
EMQILFILETYPE	281	WARNING	不正なファイル種別である
EMQILMQDBFORM	276	WARNING	ファイルが MQDB フォーマットではない
EMQILNOD	299	FATAL	データ数とチャンネル数に整合性がない
EMQILOBSMODE	282	WARNING	不正な観測モードである
EMQILPOS	275	WARNING	位置が不正である
EMQILSTATION	278	WARNING	‘‘Station’’ ヘッダ中に不正な文字がある
EMQILSTNNUM	292	FATAL	不正な観測点番号である
EMQLARGEDAY	261	WARNING	MqTime の日の数字が小さ過ぎる
EMQLARGEHOUR	262	WARNING	MqTime の時間の数字が小さ過ぎる
EMQLARGEMINUTE	263	WARNING	MqTime の分の数字が小さ過ぎる
EMQLARGEMS	265	WARNING	MqTime のマイクロセカンドの数字が小さ過ぎる
EMQLARGESECOND	264	WARNING	MqTime の秒の数字が小さ過ぎる
EMQLARGEYEAR	260	WARNING	MqTime の年の数字が大き過ぎる
EMQLONGHDR	284	WARNING	ヘッダが長過ぎる
EMQLONGINFO	271	WARNING	情報文字列がヘッダ文字列よりも長い
EMQLONGLIBV	294	FATAL	ライブラリバージョンの文字列の長さが長過ぎる
EMQLONGPSGN	296	FATAL	プログラム署名が長過ぎる
EMQLONGSTATION	277	WARNING	‘‘Station’’ ヘッダのデータ文字列が長過ぎる
EMQMEMALLOC	289	FATAL	メモリの確保に失敗した
EMQMULTINFO	272	WARNING	同じヘッダが 2 つ以上存在する
EMQNOCOMMA	274	WARNING	セパレータ ’,’ が見つからない
EMQNODATASEP	283	WARNING	ヘッダ／データ分離子が見つからない
EMQNOHEADER	285	WARNING	指定されたヘッダは存在しない
EMQNONUM	286	WARNING	数字以外の文字がある
EMQNOSEMICOLON	273	WARNING	セパレータ ’:’ が見つからない
EMQNULLPTR	287	FATAL	NULL ポインタ
EMQONEDATA	256	WARNING	文字列中にデータが 1 つしかない
EMQSHORTFILE	300	FATAL	ファイルの大きさが 0 または 1 バイトである
EMQSMALLYEAR	259	WARNING	MqTime の年の数字が小さ過ぎる
EMQSTRCPY	290	FATAL	strcpy() に失敗した
EMQSTRCAT	291	FATAL	strcat() に失敗した

マクロ名	実際の番号	エラーレベル	エラー内容
EMQZEROAMPCHNL	270	WARNING	MqAmp のチャンネル数が 0 である
EMQZEROCHANNEL	297	FATAL	チャンネル数が 0 である
EMQZERODATA	268	WARNING	データ文字列の長さが 0 である
EMQZEROHDR	266	WARNING	ヘッダ文字列の長さが 0 である
EMQZEROHDRLINE	269	WARNING	ヘッダ部の長さが 0 である
EMQZEROINFO	267	WARNING	情報文字列の長さが 0 である
EMQZEROLIBV	293	FATAL	ライブラリバージョンの文字列の長さが 0 である
EMQZERONOD	298	FATAL	データ数が 0 である
EMQZEROPSGN	295	FATAL	プログラム署名の長さが 0 である

Appendix D

MQLIB におけるグローバル変数定義

以下は、MQLIB における全グローバル変数の定義である。これらは全て、インクルード・ファイル mqstrs.h に定義されている。

変数名	変数の型	変数の内容	説明
MqLibVersion	char *	"MQ100"	ライブラリのバージョン
MqAveAmp	char *	"Average_amplitude"	情報文字列 「平均振幅」
MqChannels	char *	"Channels"	情報文字列 「チャンネル数」
MqCreDate	char *	"Data_create_date"	情報文字列 「データ作成日時」
MqDataDrift	char *	"Data_drift"	情報文字列 「データドリフト」
MqDataFormat	char *	"Data_format"	情報文字列 「データフォーマット」
MqDataType	char *	"Data_type"	情報文字列 「データ種別」
MqEndRecord	char *	"End_record"	情報文字列 「終了レコード番号」
MqFileType	char *	"File_type"	情報文字列 「ファイル種別」
MqInitMotion	char *	"Initial_motion"	情報文字列 「初動時刻」
MqMaxAmp	char *	"Maximum_amplitude"	情報文字列 「最大振幅」
MqModDate	char *	"Last_modified_date"	情報文字列 「最終修正日時」
MqNumOfData	char *	"Number_of_data"	情報文字列 「データ数」
MqObsMode	char *	"Observation_mode"	情報文字列 「観測モード」
MqOrgFile	char *	"Original_data_file"	情報文字列 「元のデータファイル名」
MqRiseTime	char *	"Rise_time"	情報文字列 「立ち上がり時刻」
MqSampRate	char *	"Sampling_rate"	情報文字列 「サンプリング間隔」
MqStartRecord	char *	"Start_record"	情報文字列 「開始レコード番号」
MqStartTime	char *	"Start_time"	情報文字列「開始時刻」
MqStation	char *	"Station"	情報文字列「観測点」
MqTapeNumber	char *	"Tape_number"	情報文字列「テープ番号」
LPFileNameExt	char *	".lp"	長周期データのファイル拡張子
SPFileNameExt	char *	".sp"	短周期データのファイル拡張子
TDLFileNameExt	char *	".tdl"	潮汐データのファイル拡張子
MqLineEnd	char *	0x0d, 0x0a, 0x00	行末記号
MqSeparator	char *	@@	ヘッダ／データ分離子
MqTopLineSeparator	char *	' '	ファイル先頭行における分離子
MqHeaderSeparator	char *	":TAB [1]"	ヘッダ部における分離子
MqDataSeparator	char *	','	データ部における分離子
MqDataTypeName	char **	{ "LP", "SP", "TIDAL" }	データ種別名
MqDataFormatName	char **	{ "OLD", "NEW" }	データフォーマット名
MqObsModeName	char **	{ "PEAKED", "FLAT" }	観測モード名
MqFileTypeName	char **	{ "FULLTEXT", "COMPOSITE" }	ファイル種別名
MqSampRateLP	double	0.15094	長周期データのサンプリング間隔
MqSampRateSP	double	0.02082	短周期データのサンプリング間隔
MqSampRateTIDAL	double	0.15094	潮汐データのサンプリング間隔

Appendix E

第３版から第４版への変更点

本書第３版から第４版への変更点は、次の通りである（主要な部分のみ)。

◎ 次の構造体が追加された。

- ☆ MqLibInfo
- ☆ MqLibVer

◎ 次の列挙型変数が追加された。

- ☆ MQAM
- ☆ MQIO
- ☆ MQRW

◎ 次の列挙型変数の名称が変更された。

- ☆ MqFormat → MQDF
- ☆ MqType → MQDT
- ☆ MqFile → MQFT

◎ 次の構造体が廃止された。

- ☆ MqError

◎ 次の関数が追加された。

- ☆ MqCheckLibVer()
- ☆ MqSetWarnLevel()
- ☆ MqReaTopLine()
- ☆ MqGetHeader()
- ☆ MqInit()

◎ 次の関数が廃止された。

◎ 次のヘッダが追加された。

◎ 次のヘッダが廃止された。

☆ 「データドリフト」ヘッダ (Data_drift)

☆ 「立ち上がり時間」ヘッダ (Rise_time)

☆ 「初動時刻」ヘッダ (Init_motion)

◎ ヘッダの追加・廃止に伴い、構造体の仕様が一部変更された。

◎ FORTRAN 用インタフェースの記述が追加された。

◎ ユーティリティ・プログラム wm2mq、testgen についての記述が追加された。