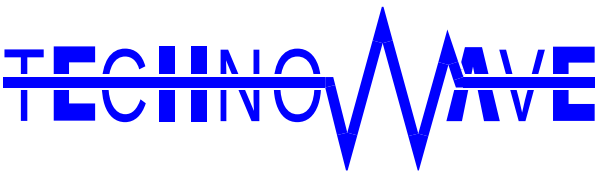
CUstation プログラミングマニュアル



テクノウェーブ株式会社

目次

1. はじめに 4

□ 専用ライブラリについて 4
□ 本マニュアルの対応ファイルとバージョンについて 4
□ マニュアル内の表記について 4

2. プログラミングの準備 5

□ C/C++での開発に必要なファイル 5
□ Visual Basic® 6.0、Visual Basic for Applications での開発に必要なファイル 6
□ Visual Basic(.NET 以降)での開発に必要なファイル 6
□ プログラムの実行時に必要なファイルについて 7

3. プログラミング方法 8

□ デバイスへの接続 8
デバイスへの接続と切断 8
デバイスを初期化する 10
通信の開始と停止 10
接続/切断、初期化の例 11
□ メモリとレジスタの操作 12
メモリ、または、レジスタからの読み出し 12
メモリ、または、レジスタへの書き込み 12
メモリとレジスタ操作の例 13
□ 割り込み通知をメッセージで受け取る 15
割り込み通知の設定 15
メッセージの受信 15
割り込み通知設定の例 16

4. 関数リファレンス 17

□ 関数の戻り値の意味 17
□ レジスタ定数 18
□ デバイスとの接続／切断および設定を行う関数 19
CU_Open() 19
CU_OpenByName() 19
CU_Close() 20
CU_Listen() 20
CU_Accept() 20
CU_CloseListenSocket() 21
CU_ReadNumber() 21

CU_SetTimeouts() *21*
CU_SetPassword() *21*
CU_SetNetworkPort() *22*
CU_GetSocket() *22*
□ 「MKY43」の基本操作関数 23
CU_Initialize() *23*
CU_Start() *23*
CU_Stop() *23*
CU_Write() *24*
CU_Read() *24*
CU_Write16() *24*
CU_Read16() *25*
CU_Write8() *25*
CU_Read8() *25*
CU_SetIntMessage() *26*
CU_ClearIntsr() *26*
□ 「MKY43」のメール関連関数 27
CU_MailSend() *27*
CU_MailReadSendStatus() *27*
CU_MailEnableReceive() *28*
CU_MailReceive() *28*
CU_MailReadReceiveStatus() *29*
□ 「MKY43」のその他の機能用関数 30
CU_Ping() *30*
CU_Query() *30*
サポート情報 31

1. はじめに

□ 専用ライブラリについて

本マニュアルでは『CUstation-USB』、および、『CUstation-LAN』に付属の専用ライブラリの使用方法と、各 API 関数の説明を行っています。このライブラリは、製品に内蔵される CUnet[1]専用 IC「MKY43」[2]のメモリや各種レジスタの操作を目的としています。パソコン上のアプリケーションプログラムはライブラリ関数を呼び出すことで CUnet によるネットワークに接続し、各ステーションとの通信を行うことができます。

本マニュアルでは CUnet プロトコルや「MKY43」についての説明は記載されませんので、これらについては株式会社ステップテクニカ発行のユーザーズマニュアルやデータシートを参照してください。

□ 本マニュアルの対応ファイルとバージョンについて

本マニュアルは下記のバージョンのファイル内容を元に記載されています。

表 1　対応するライブラリのバージョン

ファイル名	バージョン番号	対応 OS
CULIB.dll	1.3.0.x[3]	Windows® XP、Vista、7、8、8.1、10

□ マニュアル内の表記について

本マニュアル内では、対応製品『CUstation-USB』、および、『CUstation-LAN』を「デバイス」と表記する場合があります。また、『CUstation-USB』を「USB デバイス」、『CUstation-LAN』を「LAN デバイス」と表記する場合があります。

数値について「0x」、「&H」 、「H'」はいずれもそれに続く数値が 16 進数であることを表します。“0x10”、“&H1F”、“H'20”などはいずれも 16 進数です。同様に「B'」に続く数値は 2 進数であることを表します。例えば“B'01000001”のように表記されます。数値の最初に特別な表記が無い場合は 10 進数です。

---

1 CUnet は、株式会社ステップテクニカの登録商標です。

2 MKY43 は、株式会社ステップテクニカの製品です。

3 x にはより細かなバージョンを示す数値が入ります。

2. プログラミングの準備

□ C/C++での開発に必要なファイル

表 2はC/C++で開発を行うために必要なファイルです。製品付属の設定ツールをインストールした場合は、ローカルドライブにコピーが作られ、デフォルトの設定では[スタート]メニュー→[すべてのプログラム](または[プログラム])→[テクノウェーブ]→[ライブラリ]を選択して表示することができます。

表 2　C/C++での開発に必要なファイル

ファイル名	説明	付属 CD 内の格納フォルダ
CULIB.h	ライブラリを使用するためのヘッダーファイル	「¥DLL」フォルダ
CULIB.lib(32bit 用)	ライブラリを静的にリンクするための lib ファイル	「¥DLL」フォルダ
CULIB.lib(64bit 用)		「¥DLL¥X64」フォルダ

「CULIB.h」は、ライブラリの関数や定数を使用するソースファイルでインクルードしてください。

「CULIB.lib」はプロジェクトをビルドする際のリンクファイルに含める必要があります。Visual Studio® では、下のように *#pragma* を使用してソースファイル中でリンク指定することもできます。

```
#include"CULIB.h"
#pragma comment(lib, "CULIB.lib")
```

これらのファイルはコンパイラがビルド時に検索できるフォルダにコピーしておく必要があります。最も簡単な方法は、ビルドするプロジェクトと同一フォルダにコピーすることです。

複数のプロジェクトを開発する場合は、これらのファイルを格納したフォルダを、開発環境の標準のインクルードパスや標準のリンクパスに追加すると便利です。追加の方法は開発環境によって異なりますので、それぞれのオンラインヘルプなどを参照してください。

- 「CULIB.h」は WIN32 API 固有の型などを使用しています。「コンソール アプリケーション」や「フォーム アプリケーション」を作成する場合には、「CULIB.h」より前に「Windows.h」のインクルードが必要な場合があります。

□ Visual Basic® 6.0、Visual Basic for Applications での開発に必要なファイル

表 3 は Visual Basic 6.0 または Visual Basic for Applications(以下 VBA)で開発を行うために必要なファイルです。製品付属の設定ツールをインストールした場合は、ローカルドライブにコピーが作られ、デフォルトの設定では[スタート]メニュー→[すべてのプログラム](または[プログラム])→[テクノウェーブ]→[ライブラリ]を選択して表示することができます。

表 3　Visual Basic 6.0、Visual Basic for Applications での開発に必要なファイル

ファイル名	説明	付属 CD 内の格納フォルダ
CULIB.bas	ライブラリを使用するための定義ファイル	「¥DLL」フォルダ

Visual Basic 6.0 の場合、上記ファイルをプロジェクトウィンドウにドラッグ・アンド・ドロップで追加することで、ライブラリの呼び出しが可能になります。

VBA の場合、開発を行うアプリケーションソフトで [Alt] + [F11]キーを押し、Visual Basic Editor を起動し、上記ファイルをプロジェクトウィンドウにドラッグ・アンド・ドロップで追加することで、ライブラリの呼び出しが可能になります。

- VBA ではプロジェクトに追加したファイルは、ドキュメントファイル内にコピーが作成されます。ファイルを更新する場合は、以前に追加したファイルを一度解放し、新しいファイルを追加してください。

□ Visual Basic(.NET 以降)での開発に必要なファイル

表 4 は Visual Basic で開発を行うために必要なファイルです。製品付属の設定ツールをインストールした場合は、ローカルドライブにコピーが作られ、デフォルトの設定では[スタート]メニュー→[すべてのプログラム](または[プログラム])→[テクノウェーブ]→[ライブラリ]を選択して表示することができます。

表 4　Visual Basic での開発に必要なファイル

開発環境	ファイル名	説明	付属 CD 内の格納フォルダ
Visual Basic	CULIB.vb	ライブラリを使用するための定義ファイル	「¥DLL」フォルダ

Visual Studio の「ソリューション エクスプローラ」を開き、対応するファイルを開発プロジェクトの中にドラッグ・アンド・ドロップで追加することで、ライブラリの呼び出しが可能になります。これらのファイルは 32 ビット、64 ビットのどちらのプログラムを作成する場合にも共通で利用可能です。

□ プログラムの実行時に必要なファイルについて

USB デバイスでは実行に必要なファイルは、デバイスドライバのインストール時にシステムにコピーされますので、ドライバが正しくインストールされたパソコンであれば、開発したアプリケーションプログラムを実行することができます。

LAN デバイスでは実行に必要なファイルは、設定ツールのインストール時にシステムにコピーされます。設定ツールがインストールされていないパソコンで、開発したアプリケーションプログラムを実行する場合は、表 5 を参考にファイルをシステムフォルダ(通常、「¥Windows¥System32」)、もしくはアプリケーションプログラムの実行ファイル(.exe)と同一フォルダにコピーしてください。

64 ビット OS に対してファイルを手動でコピーする場合は、作成したアプリケーションプログラムが 32 ビットの命令で動作するものか、64 ビットの命令で動作するものかによって、使用される DLL とシステムフォルダの位置が変わるので注意が必要です(表 6)。

表 5　プログラムの実行に必要なファイル

ファイル名	説明	付属 CD 内の格納フォルダ	
CULIB.dll	ライブラリ本体。	32 ビット版	「¥DLL」フォルダ
		64 ビット版	「¥DLL¥X64」フォルダ
USBM3069.dll	製品を制御するための下位ライブラリ。	32 ビット版	「¥DLL」フォルダ
		64 ビット版	「¥DLL¥X64」フォルダ

表 6　必要な DLL と対応するシステムフォルダ

OS	実行するプログラム	必要な DLL	DLL のコピー先システムフォルダ
64 ビット OS	32 ビットプログラム	32 ビット版 DLL	「Windows¥SysWOW64」フォルダ
	64 ビットプログラム	64 ビット版 DLL	「Windows¥System32」フォルダ
32 ビット OS	32 ビットプログラム	32 ビット版 DLL	「Windows¥System32」フォルダ

3. プログラミング方法

以下の章では、専用ライブラリを使用して製品内蔵の「MKY43」にアクセスする方法を説明しています。本文中で使用方法を解説している *CU_* で始まるライブラリ関数については「関数リファレンス」(17 ページ以降)に詳しい説明がありますので合わせてご参照ください。

□ デバイスへの接続

専用ライブラリの各関数を使用するためには、まず制御するデバイスに接続する必要があります。接続に成功すると、そのデバイスを識別するためのハンドルを取得できますので、以降そのハンドルを使用して制御を行います。表 7 にデバイスの接続/切断時に使用する関数をあげます。

表 7　デバイスの接続/切断に使用する関数

関数名	説明
CU_Open()	デバイスに接続し、制御に必要なハンドルを取得します。
CU_OpenByName()	アドレスを指定してデバイスに接続します。LAN デバイス専用です。
CU_Close()	デバイスを切断し、制御を終了します。
CU_Listen()	クライアントモードに設定されたデバイスと接続するためのネットワークポートを作成します。LAN デバイス専用です。
CU_Accept()	クライアントモードに設定されたデバイスから接続要求があれば受け入れます。LAN デバイス専用です。
CU_CloseListenSocket()	クライアントモードに設定されたデバイスと接続するために作成したネットワークポートを閉じます。LAN デバイス専用です。
CU_Initialize()	デバイスを使用するための一般的な初期化作業を行います。
CU_Start()	SCR の START ビットをセットし、通信を開始します。
CU_Stop()	SCR の START ビットを 0 にリセットし、通信を終了します。

デバイスへの接続と切断

CU_Open() 関数を使用すると、デバイスに予め設定した装置番号を指定して接続を行うことができます。下のリストは *CU_Open()* 関数の各言語での宣言です(上から、C 言語、Visual Basic 6.0、Visual Basic .NET 以降の順です)。

```
CU_STATUS CU_Open(CU_HANDLE *phDev, DWORD Number, long Opt)
```

```
Function CU_Open(ByRef phDev As Long, ByVal Number As Long, ByVal Opt As Long) As Long
```

```
Function CU_Open(ByRef phDev As System.IntPtr,
                 ByVal Number As Integer, ByVal Opt As Integer) As Integer
```

接続に成功すると 1 番目の引数 *phDev* にハンドルが格納されます。

CU_Open() 関数の 2 番目の引数 *Number* には複数のデバイスを識別するための装置番号を指定します。装置番号は製品の付属ツールで変更することが可能です(初期値は 1 となっています)。*Number* を 0 とすると、装置番号を限定せず、最初に見つかったデバイスに接続を試みます。装置番号の設定方法は製品のハードウェアマニュアルを参照してください。

3 番目の引数 *Opt* にはデバイスのインタフェースを指定し、接続するデバイスを USB デバイス、または、LAN デバイスのみに限定することができます。*Opt* 引数の詳細は 19 ページを参照してください。

戻り値は関数の終了ステータスを表します。戻り値の意味は 17 ページを参照してください。

CU_Open() 関数で LAN デバイスと接続する場合、最初にネットワーク内の製品を検索する必要があります。アプリケーション起動後の初回の呼び出しでは必ず *Opt* 引数に *CU_LIST_UPDATE* を指定してください。このオプションが指定されると、ライブラリは検索用のパケットをブロードキャストし、接続可能なデバイスを記録する内部テーブルを更新します。

また、*CU_Open()* 関数で接続可能なのは、パソコンと同一セグメントにあるデバイスのみです。異なるネットワークに存在するデバイスには接続することができません。ルーターなどを介して、パソコンと異なるネットワークのデバイスと接続する場合には、次の *CU_OpenByName()* 関数を使用し、LAN デバイスのアドレスを指定します。引数 *pName* に IP アドレスまたはドメイン名を指定してください。

```
CU_STATUS CU_OpenByName(CU_HANDLE *phDev, LPCTSTR pName)
```

```
Function CU_OpenByName(ByRef phDev As Long, ByVal pName As String) As Long
```

```
Function CU_OpenByName(ByRef phDev As System.IntPtr, ByVal pName As String) As Integer
```

LAN デバイスは設定により、クライアントモードでの動作が可能です。クライアントモードとデフォルト動作のサーバーモードとの違いは、最初のネットワーク接続をどちらから行うかです。クライアントモードに設定したデバイスは、予め設定されたアドレスのサーバーに対して常に接続を試みます。

クライアントモードに設定された LAN デバイスからの接続を受け入れるには、*CU_Listen()* と *CU_Accept()* の 2 つの関数を使用します。

```
CU_STATUS CU_Listen(UINT_PTR *pListenSocket, LPCTSTR pLocalIP, DWORD PortNumber)
CU_STATUS CU_Accept(UINT_PTR ListenSocket, CU_HANDLE *phDev)
```

```
Function CU_Listen(ByRef pListenSocket As Long, ByVal pLocalIP As String,
                   ByVal PortNumber As Long) As Long
Function CU_Accept(ByVal ListenSocket As Long, ByRef phDev As Long) As Long
```

```
Function CU_Listen(ByRef pListenSocket As System.IntPtr, ByVal pLocalIP As String,
                   ByVal PortNumber As Integer) As Integer
Function CU_Accept(ByVal ListenSocket As System.IntPtr, ByRef phDev As System.IntPtr) As Integer
```

CU_Listen() 関数はネットワーク接続を受け入れるためのポートを準備します。この関数は成功すると *pListenSocket* 引数に一般にソケットと呼ばれる一種の識別子を返します。

次に実際にデバイスに接続するために、*CU_Accept()* 関数を呼び出します。 *ListenSocket* 引数に

は *CU_Listen()* 関数によって取得したソケットの値を渡します。関数は成功すると *phDev* 引数にデバイスを制御するためのハンドルを返します。LAN デバイスからの接続要求が無い場合は、*CU_Accept()* 関数はすぐに終了し、戻り値として *CU_DEVICE_NOT_FOUND* を返します。

ハンドルを取得した後も *CU_Listen()* 関数で取得したソケットの値は引き続き有効です。続けて *CU_Accept()* 関数を呼び出し、他の LAN デバイスからの接続要求を受け入れることができます。

CU_Listen() 関数で作成したネットワークポートを閉じる場合は、*CU_CloseListenSocket()* 関数を使用します。

1 つのデバイスを複数のプログラムから操作することはできません。他のプログラムからデバイスに接続するためには、一旦ハンドルをクローズする必要があります。ハンドルのクローズには *CU_Close()* 関数を使用します。

- ハンドルやソケットは、それを取得したプロセスが終了した場合、自動的にクローズされます。

デバイスを初期化する

デバイスを初期化するには *CU_Initialize()* 関数を使用します。宣言を以下に示します。

```
CU_STATUS CU_Initialize(CU_HANDLE hDev, long SA, long nOwn, long BpsBits, DWORD Opt)
```

```
Function CU_Initialize(ByVal hDev As Long, ByVal SA As Long, ByVal nOwn As Long,
                       ByVal BpsBits As Long, ByVal Opt As Long) As Long
```

```
Function CU_Initialize(ByVal hDev As System.IntPtr, ByVal SA As Integer, ByVal nOwn As Integer,
                       ByVal BpsBits As Integer, ByVal Opt As Integer) As Integer
```

1 番目の引数には *CU_Open()* 関数などで取得したハンドルを渡します。

2 番目の引数 *SA* 、3 番目の引数 *nOwn* には、それぞれステーションアドレスと占有幅を設定します。*nOwn* を 0 とした場合、デバイスは GMM ステーション[4]として動作します。

3 番目の引数 *BpsBits* には CUnet の転送レートを定数で指定します。

Opt 引数は、初期化オプションを指定することができます。詳細は 23 ページを参照してください。

通信の開始と停止

通信を開始するには *CU_Start()* 、停止するには *CU_Stop()* 関数を使用します。これらの関数は SCR(System Control Register)の START ビットを操作します。GMM ステーションとして動作する場合、これらの関数を呼び出す必要はありません。

[4] CUnet によるネットワークのモニタリング専用ステーション。

接続/切断、初期化の例

C 言語の例

```
CU_HANDLE hDev;

//装置番号 1 のデバイスに接続
CU_Open(&hDev, 1, CU_IF_ANY | CU_LIST_UPDATE);

if(hDev){
  CU_Initialize(hDev, 0, 2, CU_BPS_12M, 0);  //SA=0,OWN=2 でデバイスを初期化
  CU_Start(hDev); //通信を開始

  /* 必要な操作を記述します */

  CU_Stop(hDev);  //通信を終了
  CU_Close(hDev);  //デバイスを閉じる
}
```

VisualBasic 6.0 の例

```
Dim hDev As Long

'装置番号 1 のデバイスに接続
CU_Open hDev, 1, CU_IF_ANY Or CU_LIST_UPDATE

If hDev <> 0 Then
  CU_Initialize hDev, 0, 2, CU_BPS_12M, 0  'SA=0,OWN=2 でデバイスを初期化
  CU_Start hDev '通信を開始

  '必要な操作を記述します

  CU_Stop hDev '通信を終了
  CU_Close hDev  'デバイスを閉じる
End If
```

VisualBasic .NET 以降の例

```
Dim hDev As System.IntPtr

'装置番号 1 のデバイスに接続
CU_Open(hDev, 1, CU_IF_ANY Or CU_LIST_UPDATE)

If hDev <> 0 Then
  CU_Initialize(hDev, 0, 2, CU_BPS_12M, 0) 'SA=0,OWN=2 でデバイスを初期化
  CU_Start(hDev) '通信を開始

  '必要な操作を記述します

  CU_Stop(hDev)  '通信を終了
  CU_Close(hDev) 'デバイスを閉じる
End If
```

□ メモリとレジスタの操作

グローバルメモリ、メールの送受信バッファ、制御用レジスタは全て同じ方法でアクセスすることができます。表 8 にこれらの操作に使用する関数をあげます。

表 8　メモリとレジスタの操作に使用する関数

関数名	説明
CU_Read()	複数バイトのデータを「MKY43」のメモリまたはレジスタから読み出します。
CU_Write()	複数バイトのデータを「MKY43」のメモリまたはレジスタに書き込みます。
CU_Read8()	1 バイトのデータを「MKY43」のメモリまたはレジスタから読み出します。
CU_Write8()	1 バイトのデータを「MKY43」のメモリまたはレジスタに書き込みます。
CU_Read16()	1 ワード(16 ビット)のデータを「MKY43」のメモリまたはレジスタから読み出します。
CU_Write16()	1 ワード(16 ビット)のデータを「MKY43」のメモリまたはレジスタに書き込みます。

メモリ、または、レジスタからの読み出し

「MKY43」のメモリ、または、レジスタから読み出しを行う場合、*CU_Read8()*、*CU_Read16()*、*CU_Read()* 関数を使用します。*CU_Read8()*、*CU_Read16()* 関数はそれぞれ、8 ビットデータ、16 ビットデータを読み出します。*CU_Read()* 関数は指定バイト数のデータを読み出します。アドレスは「MKY43」のユーザーズマニュアルに記載のアドレスを指定します。レジスタにアクセスする場合は、各言語用の定義ファイルに用意された定数(18 ページ)で指定することもできます。

メモリ、または、レジスタへの書き込み

「MKY43」のメモリ、または、レジスタに書き込みを行う場合には、*CU_Write8()*、*CU_Write16()*、*CU_Write()* 関数を使用します。読み出し同様、それぞれ 8 ビット、16 ビット、指定バイト数の書き込みを行います。

CU_Write8()、*CU_Write16()* 関数については、マスク用の引数を指定して書き込みビットを指定することができます。*CU_Write8()* と *CU_Write16()* 関数の宣言を示します。

```
CU_STATUS CU_Write8(CU_HANDLE hDev, DWORD Address, BYTE bData, BYTE bMask)
CU_STATUS CU_Write16(CU_HANDLE hDev, DWORD Address, WORD wData, WORD wMask)
```

```
Function CU_Write8(ByVal hDev As Long, ByVal Address As Long, ByVal bData As Byte,
                   ByVal bMask As Byte) As Long
Function CU_Write16(ByVal hDev As Long, ByVal Address As Long, ByVal wData As Integer,
                    ByVal wMask As Integer) As Long
```

```
Function CU_Write8(ByVal hDev As System.IntPtr, ByVal Address As Integer, ByVal bData As Byte,
                   ByVal bMask As Byte) As Integer
Function CU_Write16(ByVal hDev As System.IntPtr, ByVal Address As Integer, ByVal wData As Short,
                    ByVal wMask As Short) As Integer
```

bMask、*wMask* 引数はマスク値となっています。この値が 1 のビットだけが書き込みの対象となり、0 のビットは無視されます。図 1 は *CU_Write8()* 関数を使用して対象アドレスの下位 4 ビットのみ書き

換える例です。

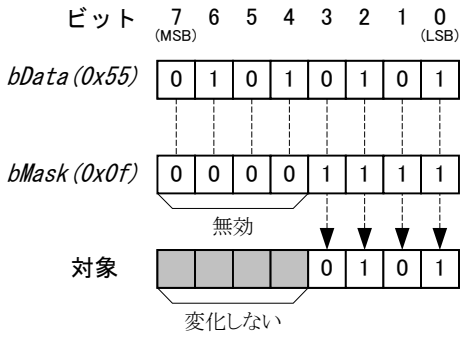


図 1　書き込みのマスク

メモリとレジスタ操作の例

以下は、メモリやレジスタ操作の例です(デバイスの接続/切断、初期化などの操作は省略されています)。

C 言語の例

```
BYTE st;
WORD ssr;
BYTE gm[512];
BYTE write_data[16];
int i;

for(i=0;i<16;i++) write_data[i] = i;

//ステーションタイムの読み出し
CU_Read8(hDev, CU_REG_SCR, &st);

//システムステータスレジスタの読み出し
CU_Read16(hDev, CU_REG_SSR, &ssr);

//グローバルメモリの読み出し
CU_Read(hDev, 0, gm, 512);

//グローバルメモリへの書き込み
CU_Write(hDev, 0, write_data, 16);
```

VisualBasic 6.0 の例

```
Dim st As Byte
Dim ssr As Integer
Dim gm(511) As Byte
Dim write_data(15) As Byte
Dim i As Integer;

For i = 0 To 15
    write_data(i) = i
Next

'ステーションタイムの読み出し
CU_Read8 hDev, CU_REG_SCR, st

'システムステータスレジスタの読み出し
CU_Read16 hDev, CU_REG_SSR, ssr

'グローバルメモリの読み出し
CU_Read hDev, 0, gm(0), 512

'グローバルメモリへの書き込み
CU_Write hDev, 0, write_data(0), 16
```

VisualBasic .NET 以降の例

```
Dim st As Byte
Dim ssr As Short
Dim gm(511) As Byte
Dim write_data(15) As Byte
Dim i As Integer;

For i = 0 To 15
    write_data(i) = i
Next

'ステーションタイムの読み出し
CU_Read8(hDev, CU_REG_SCR, st)

'システムステータスレジスタの読み出し
CU_Read16(hDev, CU_REG_SSR, ssr)

'グローバルメモリの読み出し
CU_Read(hDev, 0, gm, 512)

'グローバルメモリへの書き込み
CU_Write(hDev, 0, write_data, 16)
```

□ 割り込み通知をメッセージで受け取る

INT0SR、INT1SR などの割り込みフラグが変化した際に Windows のメッセージで通知を受けることができます。

表 9 メッセージによる割り込み通知に使用する関数

関数名	説明
CU_SetIntMessage()	メッセージ番号、通知先などの設定を行います。
CU_ClearIntsr()	割り込みフラグのクリアを行います。

割り込み通知の設定

割り込み発生を知らせるメッセージを受け取るには、まず *CU_SetIntMessage()* 関数を呼び出します。関数の宣言を以下に示します。

```
CU_STATUS CU_SetIntMessage(CU_HANDLE hDev, UINT Message, HWND hWnd, DWORD idThread)
```

```
Function CU_SetIntMessage(ByVal hDev As Long, ByVal Message As Long,
                          ByVal hWnd As Long, ByVal idThread As Long) As Long
```

```
CU_SetIntMessage(ByVal hDev As System.IntPtr, ByVal Message As Integer,
                 ByVal hWnd As System.IntPtr, ByVal idThread As Integer) As Integer
```

Message 引数に受け取るメッセージ番号を指定します。通常は WM_APP(0x8000)～0xbfff の範囲で指定します。

メッセージの通知先は、ウィンドウかスレッド(または両方)を指定できます。ウィンドウで受け取る場合は *hWnd* 引数にウィンドウハンドルを指定します。スレッドで受け取る場合には *idThread* 引数にスレッド ID を指定します。

割り込みの通知を受けたい要因を指定するには *CU_Write16()* 関数を使用して INT0CR、または、INT1CR を設定します。

メッセージの受信

INT0CR、または、INT1CR に指定した割り込み要因が実際に発生すると、*CU_SetIntMessage()* 関数で指定した番号のメッセージが、ウィンドウ、または、スレッドにポストされます。その際、メッセージのパラメータとして渡される *wParam* はそのメッセージが INT0SR の変化によるものか INT1SR の変化によるものかを 0、または、1 で示します。*lParam* は INT0SR(INT1SR)同様に割り込み要因を示します。

同じ要因によるメッセージが繰り返しポストされるのを防ぐために、一度通知された割り込み要因は、メッセージによる通知が禁止されます。通知を再度許可するために、割り込みステータスのクリアには *CU_ClearIntsr()* 関数を使用してください。*CU_Write16()* 関数などで直接 INT0SR や INT1SR のビットをクリアすると以降その割り込み要因によるメッセージが発生しなくなります。

```
CU_STATUS CU_ClearIntsr(CU_HANDLE hDev, long Ch, DWORD ClrBits)
```

```
Function CU_ClearIntsr(ByVal hDev As Long, ByVal Ch As Long, ByVal ClrBits As Long) As Long
```

```
Function CU_ClearIntsr(ByVal hDev As System.IntPtr, ByVal Ch As Integer,
                       ByVal ClrBits As Integer) As Integer
```

Ch 引数は INT0SR をクリアする場合には 0 を、INT1SR をクリアする場合には 1 を指定します。
ClrBits 引数は INT0SR、または、INT1SR のクリアしたいビットを 1、その他のビットは 0 とします。

割り込み通知設定の例

以下は、割り込み通知設定の例です。(デバイスの接続/切断、初期化などの操作は省略されています)。メッセージハンドラの記述などについては付属のサンプルプログラムを参照してください。

C 言語の例

```
//メッセージの設定
CU_SetIntMessage(hDev, WM_APP + 1, hWnd, 0); //hWnd にメッセージを受け取るウィンドウを指定

//DR(Data Renewal)の割り込みを許可
CU_Write16(hDev, CU_REG_INT0CR, 0x0002);

//マスクを確実に解除するため全ての割り込みステータスをリセット
CU_ClearIntsr(hDev, 0, 0xffff);
```

VisualBasic 6.0 の例

```
'メッセージの設定
CU_SetIntMessage hDev, WM_APP + 1, hWnd, 0 'hWnd にメッセージを受け取るウィンドウを指定

'DR(Data Renewal)の割り込みを許可
CU_Write16 hDev, CU_REG_INT0CR, &H2

'マスクを確実に解除するため全ての割り込みステータスをリセット
CU_ClearIntsr hDev, 0, &HFFFF
```

VisualBasic .NET 以降の例

```
'メッセージの設定
CU_SetIntMessage(hDev, WM_APP + 1, hWnd, 0) 'hWnd にメッセージを受け取るウィンドウを指定

'DR(Data Renewal)の割り込みを許可
CU_Write16(hDev, CU_REG_INT0CR, &H2)

'マスクを確実に解除するため全ての割り込みステータスをリセット
CU_ClearIntsr(hDev, 0, &HFFFF)
```

4. 関数リファレンス

各関数の説明は、C 言語、Visual Basic 6.0 以前、Visual Basic.NET 以降、それぞれにおける関数宣言、引数の説明、動作説明の順になっています。

関数の戻り値は 32 ビットの整数で関数の実行結果を表します（以下参照）。

□ 関数の戻り値の意味

以下に主な戻り値の意味を示します。尚、戻り値を示す各定数は各言語用の定義ファイル（拡張子が「.h」、「.bas」、「.vb」のファイル）中で定義されています。

表 10　関数の戻り値

定数	値	意味
CU_OK	0x00000000	正常終了
CU_INVALID_HANDLE	0x00000001	デバイスのハンドルが無効
CU_DEVICE_NOT_FOUND	0x00000002	デバイスが見つからない
CU_IO_ERROR	0x00000004	送受信中にエラーが発生した
CU_INSUFFICIENT_RESOURCES	0x00000005	リソースエラー(デバイスの最大接続数を超えた場合など)
CU_INVALID_ARGS	0x00000010	関数に渡された引数が無効
CU_NOT_SUPPORTED	0x00000011	サポートされない機能
CU_OTHER_ERROR	0x00000012	その他のエラー
CU_TIMEOUT	0xffff0001	送信または受信処理がタイムアウトした
CU_FILE_ERROR	0xffff0002	ファイル操作に関するエラーが発生した
CU_MEMORY_ERROR	0xffff0003	メモリの確保に失敗した
CU_DATA_NOT_FOUND	0xffff0004	有効なデータが見つからなかった
CU_SOCKET_ERROR	0xffff0005	Winsock のエラー（多くの場合 WSAGetLastError() を呼び出すとさらに詳しい情報を得ることができます）
CU_ACCESS_DENIED	0xffff0006	デバイスとの認証作業に失敗した
CU_NOT_SUPPORTED_MODE	0xffff0007	関数をサポートしないモードでデバイスに接続している
CU_FLASH_MODE_DEVICE	0xffff0008	フラッシュ書換えモードのデバイスのため制御できない
CU_MAIL_SENDING	0xffffff01	メール送信中のため新たな送信を開始できない。
CU_TRYING_QUERY	0xffffff02	クエリ実行中のため、PING を送信できない。
CU_QUERY_TIMEOUT	0xffffff03	クエリに対する待ち時間が経過した。

□ レジスタ定数

表 11　レジスタアドレスを示す定数

定数	値	意味
CU_REG_RFR	0x300	RFR(Receive Flag Register)
CU_REG_LFR	0x308	LFR(Link Flag Register)
CU_REG_MFR	0x310	MFR(Member Flag Register)
CU_REG_DRFR	0x318	DRFR(Data Renewal Flag Register)
CU_REG_LGR	0x320	LGR(Link Group Register)
CU_REG_MGR	0x328	MGR(Member Group Register)
CU_REG_DRCR	0x330	DRCR(Data Renewal Check Register)
CU_REG_RHCR0	0x338	RHCR0(Read Hazard Control Register 0)
CU_REG_RHCR1	0x33a	RHCR1(Read Hazard Control Register 1)
CU_REG_WHCR0	0x33c	WHCR0(Write Hazard Control Register 0)
CU_REG_WHCR1	0x33e	WHCR1(Write Hazard Control Register 1)
CU_REG_MSLR	0x340	MSLR(Mail Send Limit time Register)
CU_REG_MSRR	0x342	MSRR(Mail Send Result Register)
CU_REG_MESR	0x344	MESR(Mail Error Status Register)
CU_REG_MSCR	0x346	MSCR(Mail Send Control Register)
CU_REG_MR0CR	0x348	MR0CR(Mail Receive 0 Control Register)
CU_REG_MR1CR	0x34a	MR1CR(Mail Receive 1 Control Register)
CU_REG_CCTR	0x34c	CCTR(Care CounTer Register)
CU_REG_UTCR	0x34e	UTCR(UTility pin Control Register)
CU_REG_QCR	0x350	QCR(Query Control Register)
CU_REG_NFSR	0x352	NFSR(New Final Station Register)
CU_REG_FSR	0x354	FSR(Final Station Register)
CU_REG_BCR	0x356	BCR(Basic Control Register)
CU_REG_INT0CR	0x358	INT0CR(INTerrupt 0 Control Register)
CU_REG_INT1CR	0x35a	INT1CR(INTerrupt 1 Control Register)
CU_REG_IT0CR	0x35c	IT0CR(Interrupt Timing 0 Control Register)
CU_REG_IT1CR	0x35e	IT1CR(Interrupt Timing 1 Control Register)
CU_REG_INT0SR	0x360	INT0SR(INTerrupt 0 Status Register)
CU_REG_INT1SR	0x362	INT1SR(INTerrupt 1 Status Register)
CU_REG_SSR	0x364	SSR(System Status Register)
CU_REG_SCR	0x366	SCR(System Control Register)
CU_REG_CCR	0x368	CCR(Chip Code Register)
CU_REG_RHPB0	0x370	RHPB0(Read Hazard Protection Buffer 0)
CU_REG_RHPB1	0x378	RHPB1(Read Hazard Protection Buffer 1)
CU_REG_WHPB0	0x380	WHPB0(Write Hazard Protection Buffer 0)
CU_REG_WHPB1	0x388	WHPB1(Write Hazard Protection Buffer 1)

□ デバイスとの接続／切断および設定を行う関数

CU_Open()

```
CU_STATUS CU_Open(CU_HANDLE *phDev, DWORD Number, long Opt)
Function CU_Open(ByRef phDev As Long, ByVal Number As Long, ByVal Opt As Long) As Long
Function CU_Open(ByRef phDev As System.IntPtr,
                 ByVal Number As Integer, ByVal Opt As Integer) As Integer
```

phDev : [出力]デバイスへのハンドルの格納先
Number : 接続するデバイスの装置番号
Opt : 接続オプション。以下の組み合わせ
CU_IF_LAN(0x40000000) : LAN デバイスに接続する
CU_IF_USB(0x20000000) : USB デバイスに接続する
CU_IF_ANY(0xff000000) : インタフェースを問わない

上記インタフェースに加えて以下のオプションを OR で追加可能
CU_LIST_UPDATE(0x00000001) : LAN デバイスの情報を保存した内部テーブルを更新
(LAN デバイスのみ有効)

Number に指定された装置番号の CUnet インタフェースデバイスと接続します。
Number が 0 の場合、最初に見つかったデバイスに接続を行います。
Opt は CU_IF_USB または CU_IF_LAN でインタフェースを指定します。これらは OR で結合することが可能です。

CU_IF_LAN を指定した場合には CU_LIST_UPDATE を結合することができます。このオプションにより LAN デバイスの情報を格納した内部テーブルが更新されます。LAN デバイスに接続する場合、内部テーブルに情報を取得するため、少なくとも 1 回はこのオプションを指定する必要があります。CU_LIST_UPDATE が指定されない場合には、既に取得した内部テーブルの情報からデバイスを検索しますので、追加されたデバイスが無いことが明らかであれば、省略することが可能です。
内部テーブルの更新には UDP のブロードキャストが用いられ、約 1.5 秒の時間を要します。

USB と LAN の両方のインタフェースが指定された場合、まず USB デバイスに対して接続を試み、接続ができなかった場合に LAN デバイスへの接続を行います。

CU_OpenByName()

```
CU_STATUS CU_OpenByName(CU_HANDLE *phDev, LPCTSTR pName)
Function CU_OpenByName(ByRef phDev As Long, ByVal pName As String) As Long
Function CU_OpenByName(ByRef phDev As System.IntPtr, ByVal pName As String) As Integer
```

phDev : [出力]デバイスへのハンドルの格納先
pName : デバイスの IP アドレス、または、ドメイン名

CUstation-LAN 専用の関数です。IP アドレス、または、ドメイン名を指定してデバイスに接続します。異なるネットワークやインターネットを介しての接続に使用します。
ポート番号も指定する場合は、"192.168.10.10:49152"のようにアドレスに続けて":"(コロン)、ポート番号の順で指定します。

CU_Close()

```
CU_STATUS CU_Close(CU_HANDLE hDev)
Function CU_Close(ByVal hDev As Long) As Long
Function CU_Close(ByVal hDev As System.IntPtr) As Integer
```

hDev : デバイスのハンドル

デバイスの操作を終了します。ハンドルは無効になります。

CU_Listen()

```
CU_STATUS CU_Listen(UINT_PTR *pListenSocket, LPCTSTR pLocalIP, DWORD PortNumber)
Function CU_Listen(ByRef pListenSocket As Long,
                   ByVal pLocalIP As String, ByVal PortNumber As Long ) As Long
Function CU_Listen(ByRef pListenSocket As System.IntPtr,
                   ByVal pLocalIP As String, ByVal PortNumber As Integer) As Integer
```

pListenSocket : [出力]接続待ちポートを識別するソケットの格納先
pLocalIP : [入力]接続待ちを行うローカル側(パソコン)の IP アドレス
PortNumber : 接続待ちを行うポート番号

指定のポート番号をオープンし、クライアントモードのデバイスの接続を待ちます。
pListenSocket に返された値を CU_Accept() に渡すことで、ここで設定したポートへの接続を受け入れることができます。
pLocalIP はパソコンが複数のネットワークに接続されている場合、接続待ちを行う IP アドレスの指定を行います。通常は、NULL(または vbNullString)としてください。
PortNumber はオープンする TCP ポートの番号を指定します。ネットワーク設定ツールでデバイスに設定した[サーバーポート]の番号と一致する必要があります。他のアプリケーションやサービスで使用されている番号と重なると関数は失敗し、CU_SOCKET_ERROR が返ります。

CU_Accept()

```
CU_STATUS CU_Accept(UINT_PTR ListenSocket, CU_HANDLE *phDev)
Function CU_Accept(ByVal ListenSocket As Long, ByRef phDev As Long) As Long
Function CU_Accept(ByVal ListenSocket As System.IntPtr, ByRef phDev As System.IntPtr) As Integer
```

ListenSocket : CU_Listen()関数で取得したソケットの値
phDev : [出力]デバイスへのハンドルの格納先

CU_Listen()関数でオープンした接続待ちソケットに対して、接続要求があれば受け入れてデバイスへのハンドルを返します。
接続要求が無い場合は CU_DEVICE_NOT_FOUND を返します。接続待ちを行う間はこの関数を繰り返し呼び出して、接続要求の有無を確認してください。

CU_CloseListenSocket()

```
CU_STATUS CU_CloseListenSocket(UINT_PTR ListenSocket)
Function CU_CloseListenSocket(ByVal ListenSocket As Long) As Long
Function CU_CloseListenSocket(ByVal ListenSocket As System.IntPtr) As Integer
```

ListenSocket : CU_Listen()関数で取得したソケットの値

CU_Listen()でオープンした接続待ちのソケットをクローズします。

CU_ReadNumber()

```
CU_STATUS CU_ReadNumber(CU_HANDLE hDev, DWORD *pNumber)
Function CU_ReadNumber(ByVal hDev As Long, ByRef pNumber As Long) As Long
Function CU_ReadNumber(ByVal hDev As System.IntPtr, ByRef pNumber As Integer) As Integer
```

hDev : デバイスのハンドル
pNumber : [出力]装置番号の格納先

接続中のデバイスの装置番号を読み出します。

CU_SetTimeouts()

```
CU_STATUS CU_SetTimeouts(CU_HANDLE hDev, DWORD ReadTimeout, DWORD WriteTimeout)
Function CU_SetTimeouts(ByVal hDev As Long,
                        ByVal ReadTimeout As Long, ByVal WriteTimeout As Long) As Long
Function CU_SetTimeouts(ByVal hDev As System.IntPtr,
                        ByVal ReadTimeout As Integer, ByVal WriteTimeout As Integer) As Integer
```

hDev : デバイスのハンドル
ReadTimeout : 読出しのタイムアウト時間(msec 単位)
WriteTimeout : 書込みのタイムアウト時間(msec 単位)

デバイスからデータの読出し、デバイスへの書込みの際のタイムアウト時間を設定します。デフォルトはどちらも 5 秒に設定されています。

CU_SetPassword()

```
CU_STATUS CU_SetPassword(LPCTSTR pPass)
Function CU_SetPassword(ByVal pPass As String) As Long
Function CU_SetPassword(ByVal pPass As String) As Integer
```

pPass : [入力]パスワード文字列

LAN デバイスと接続する際のパスワードを変更します。
パスワードは製品のネットワーク設定ツールでデバイスに書き込んだものと同じものを指定してください。設定ツールで変更していない場合は、呼び出しの必要はありません。
pPass を Null 値、または、 ""(空文字列)とした場合、デフォルトのパスワードが使用されます。

CU_SetNetworkPort()

CU_STATUS CU_SetNetworkPort(DWORD PortNumber)

Function CU_SetNetworkPort(ByVal PortNumber As Long) As Long

Function CU_SetNetworkPort(ByVal PortNumber As Integer) As Integer

PortNumber : デバイスとの接続に使用するポート番号

サーバーモードの LAN デバイスと接続する際に使用するデフォルトのポート番号を変更します。この関数でポート番号を変更した場合、以後のそのプロセスからの接続は全て指定番号をデフォルト値として行われます。
通常、ポート番号はネットワーク設定ツールの[ポート]に指定した値と一致している必要があります。
初期値は 49152 です。

CU_GetSocket()

UINT_PTR CU_GetSocket(CU_HANDLE hDev)

Function CU_GetSocket(ByVal hDev As Long) As Long

Function CU_GetSocket(ByVal hDev As System.IntPtr) As System.IntPtr

hDev : デバイスのハンドル

戻り値としてハンドルに結びついた SOCKET を返します。LAN デバイスでは無い場合や無効なハンドルが渡された場合は CU_INVALID_SOCKET として定義された定数が返ります。

□ 「MKY43」の基本操作関数

CU_Initialize()

```
CU_STATUS CU_Initialize(CU_HANDLE hDev, long SA, long nOwn, long BpsBits, DWORD Opt)
Function CU_Initialize(ByVal hDev As Long, ByVal SA As Long, ByVal nOwn As Long,
                       ByVal BpsBits As Long, ByVal Opt As Long) As Long
Function CU_Initialize(ByVal hDev As System.IntPtr, ByVal SA As Integer, ByVal nOwn As Integer,
                       ByVal BpsBits As Integer, ByVal Opt As Integer) As Integer
```

hDev　: デバイスのハンドル
SA　: 占有するステーションアドレス(0-63)
nOwn　: 占有するブロック数を設定(0-63)
BpsBits : ビットレート指定
　CU_BPS_12M(0xc0) : 12Mbps
　CU_BPS_6M(0x80)　: 6Mbps
　CU_BPS_3M(0x40)　: 3Mbps

Opt　: 初期化オプション
　CU_DISABLE_GM_CLEAR(0x0001)　　: GM をクリアしない
　CU_EXCLUDE_RUNNING_DEVICE(0x0002) : CUnet 接続中のデバイスは初期化しない
　CU_LONG_FRAME(0x0004)　　: フレーム長定数を 256 にします(HUB 対応)

内蔵する「MKY43」を初期化します。nOwn に 0 を指定すると GMM モードに設定されます。初期化作業は以下の内容です。

1. GM, MSB, MRB を 0 にクリア(オプションによって GM はクリアされない場合があります)
2. SCR の START ビットをクリア
3. SCR の GMM ビットに 1 をセット
4. nOwn が 0 以外の場合、BCR にステーションアドレス、ビットレート、占有幅を設定
   nOwn が 0 の場合、BCR にビットレートを設定
5. nOwn が 0 以外の場合、SCR の GMM ビットをクリア

CU_Start()

```
CU_STATUS CU_Start(CU_HANDLE hDev)
Function CU_Start(ByVal hDev As Long) As Long
Function CU_Start(ByVal hDev As System.IntPtr) As Integer
```

hDev : デバイスのハンドル

SCR の START ビットをセットします。GMM モードでは使用しません。

CU_Stop()

```
CU_STATUS CU_Stop(CU_HANDLE hDev)
Function CU_Stop(ByVal hDev As Long) As Long
Function CU_Stop(ByVal hDev As System.IntPtr) As Integer
```

hDev : デバイスのハンドル

SCR の START ビットを 0 にリセットします。GMM モードでは使用しません。

CU_Write()

```
CU_STATUS CU_Write(CU_HANDLE hDev, DWORD Address, void *pData, long nData)
Function CU_Write(ByVal hDev As Long, ByVal Address As Long, ByRef pData As Any,
                  ByVal nData As Long) As Long
Function CU_Write(ByVal hDev As System.IntPtr, ByVal Address As Integer, ByVal pData() As Byte[5],
                  ByVal nData As Integer) As Integer
```

hDev : デバイスのハンドル
Address : アドレス
pData : [入力]書き込むデータ
nData : データのバイト数

「MKY43」の指定アドレスにデータを書き込みます。

CU_Read()

```
CU_STATUS CU_Read(CU_HANDLE hDev, DWORD Address, void *pData, long nData)
Function CU_Read(ByVal hDev As Long, ByVal Address As Long, ByRef pData As Any,
                 ByVal nData As Long) As Long
Function CU_Read(ByVal hDev As System.IntPtr, ByVal Address As Integer, ByVal pData() As Byte[5],
                 ByVal nData As Integer) As Integer
```

hDev : デバイスのハンドル
Address : アドレス
pData : [出力]読み出したデータの格納先
nData : データのバイト数

「MKY43」の指定アドレスからデータを読み出します。

CU_Write16()

```
CU_STATUS CU_Write16(CU_HANDLE hDev, DWORD Address, WORD wData, WORD wMask)
Function CU_Write16(ByVal hDev As Long, ByVal Address As Long, ByVal wData As Integer,
                    ByVal wMask As Integer) As Long
Function CU_Write16(ByVal hDev As System.IntPtr, ByVal Address As Integer, ByVal wData As Short,
                    ByVal wMask As Short) As Integer
```

hDev : デバイスのハンドル
Address : アドレス
wData : 書き込むデータ
wMask : マスク(0 のビットは操作されません)

「MKY43」の指定アドレスに 16 ビットデータを書き込みます。
マスクは読み出したデータを書き戻すことで機能しますので、INT0SR や INT1SR のように書込み値と読み出し値が一致しない場合は使用できません。

[5] Short と Integer のオーバーロードが定義されています。

CU_Read16()

```
CU_STATUS CU_Read16(CU_HANDLE hDev, DWORD Address, WORD *pwData)
Function CU_Read16(ByVal hDev As Long, ByVal Address As Long, ByRef pwData As Integer) As Long
Function CU_Read16(ByVal hDev As System.IntPtr, ByVal Address As Integer,
                   ByRef pwData As Short) As Integer
```

hDev : デバイスのハンドル
Address : アドレス
pwData : [出力]データの格納先

「MKY43」の指定アドレスから 16 ビットデータを読み出します。

CU_Write8()

```
CU_STATUS CU_Write8(CU_HANDLE hDev, DWORD Address, BYTE bData, BYTE bMask)
Function CU_Write8(ByVal hDev As Long, ByVal Address As Long, ByVal bData As Byte,
                   ByVal bMask As Byte) As Long
Function CU_Write8(ByVal hDev As System.IntPtr, ByVal Address As Integer, ByVal bData As Byte,
                   ByVal bMask As Byte) As Integer
```

hDev : デバイスのハンドル
Address : アドレス
bData : 書き込むデータ
bMask : マスク(0 のビットは操作されません)

「MKY43」の指定アドレスに 8 ビットデータを書き込みます。
マスクは読み出したデータを書き戻すことで機能しますので、INT0SR や INT1SR のように書込み値と読み出し値が一致しない場合は使用できません。

CU_Read8()

```
CU_STATUS CU_Read8(CU_HANDLE hDev, DWORD Address, BYTE *pbData)
Function CU_Read8(ByVal hDev As Long, ByVal Address As Long, ByRef pbData As Byte) As Long
Function CU_Read8(ByVal hDev As System.IntPtr, ByVal Address As Integer,
                  ByRef pbData As Byte) As Integer
```

hDev : デバイスのハンドル
Address : アドレス
pbData : [出力]データの格納先

「MKY43」の指定アドレスから 8 ビットデータを読み出します。

CU_SetIntMessage()

```
CU_STATUS CU_SetIntMessage(CU_HANDLE hDev, UINT Message, HWND hWnd, DWORD idThread)
Function CU_SetIntMessage(ByVal hDev As Long, ByVal Message As Long,
                          ByVal hWnd As Long, ByVal idThread As Long) As Long
Function CU_SetIntMessage(ByVal hDev As System.IntPtr, ByVal Message As Integer,
                          ByVal hWnd As System.IntPtr, ByVal idThread As Integer) As Integer
```

hDev : デバイスのハンドル
Message : 受け取りたいメッセージ番号
hWnd : メッセージを受け取るウィンドウ
idThread : メッセージを受け取るスレッド

INT0SR、または、INT1SR のビットがセットされたときにメッセージを受け取れるように設定します。
Message には通常 WM_APP 以上の値を指定します。Message を 0 とするとメッセージによる通知を終了します。

hWnd が指定されると指定ウィンドウにメッセージがポストされます。ウィンドウへの通知が必要ない場合、NULL または 0 としてください。
idThread が指定されると指定スレッドにメッセージがポストされます。スレッドへの通知が必要ない場合、0 としてください。

メッセージハンドラに渡される wParam は 0 または 1 となり INT0SR、INT1SR のどちらに起因するメッセージかを示します。lParam は INT0SR、または、INT1SR の内容です。

同じ要因で多重にメッセージがポストされるのを防ぐために、一度通知された要因を示すビットは通知されないようにマスクされます(lParam の内容もマスクされます)。
マスクを解除するために INT0SR、または、INT1SR のクリアには CU_ClearIntsr()を呼び出してください。
レジスタを直接操作するとマスクは解除されませんので、以降その要因ではメッセージがポストされなくなります。

デバイスとの通信に要する時間があるため、全ての割り込み要因をハンドリングできるとは限りません。
メッセージが通知されてから、割り込みステータスをクリアするまでに同一の割り込み要因が発生した場合、その割り込み要因は無視されます。

CU_ClearIntsr()

```
CU_STATUS CU_ClearIntsr(CU_HANDLE hDev, long Ch, DWORD ClrBits)
Function CU_ClearIntsr(ByVal hDev As Long, ByVal Ch As Long, ByVal ClrBits As Long) As Long
Function CU_ClearIntsr(ByVal hDev As System.IntPtr, ByVal Ch As Integer,
                       ByVal ClrBits As Integer) As Integer
```

hDev : デバイスのハンドル
Ch : クリアする割り込みステータスレジスタの番号
0 : INT0SR
1 : INT1SR

ClrBits : クリアするビットを 1 とする。各ビットの意味は INT0SR/INT1SR と同様です。

指定された割り込みステータスレジスタの指定ビットをクリアします。

□ 「MKY43」のメール関連関数

メール操作に関する関数です。メール機能は GMM モードでは利用できません。

CU_MailSend()

```
CU_STATUS CU_MailSend(CU_HANDLE hDev, long SA, void *pData, long nData)
Function CU_MailSend(ByVal hDev As Long, ByVal SA As Long,
                     ByRef pData As Any, ByVal nData As Long) As Long
Function CU_MailSend (ByVal hDev As System.IntPtr, ByVal SA As Integer, ByVal pData() As Byte[6],
                      ByVal nData As Integer) As Integer
```

hDev : デバイスのハンドル
SA : 送信先のステーションアドレス
pData : [入力]送信データ
nData : 送信データのバイト数(1-256)

指定のステーションにメールを送信します。前回のメール送信が終了していない場合は、戻り値として CU_MAIL_SENDING が返ります。
実際の送信は 8 バイト単位で行われますので、nData が 8 の倍数でない場合、8 の倍数になるように切り上げられます。例えば nData が 1 の場合、実際には 8 バイトが送信されます。

CU_MailReadSendStatus()

```
CU_STATUS CU_MailReadSendStatus(CU_HANDLE hDev, DWORD *pStatus, long *pCycle)
Function CU_MailReadSendStatus(ByVal hDev As Long, ByRef pStatus As Long,
                               ByRef pCycle As Long) As Long
Function CU_MailReadSendStatus(ByVal hDev As System.IntPtr, ByRef pStatus As Integer,
                               ByRef pCycle As Integer) As Integer
```

hDev : デバイスのハンドル
pStatus : [出力]メール送信のステータス。下の値の組み合わせ
CU_MAILS_NORDY(0x01) 送信先が受信許可されていない
CU_MAILS_NOEX (0x02) 送信先ステーションが存在しない
CU_MAILS_TOUT (0x04) タイムアウト
CU_MAILS_SZFLT(0x08) 送信サイズ不正
CU_MAILS_LMFLT(0x10) タイムアウト値が不正
CU_MAILS_STOP (0x20) メール送信中のネットワーク停止
CU_MAILS_SEND (0x40) 送信中
CU_MAILS_ERR (0x80) エラー発生
pCycle : [出力]メール送信に要したサイクル数

メールの送信ステータスを取得します。
送信が正常に完了した場合は pStatus には 0 が格納され、pCycle に送信に要したサイクル数が格納されます。
まだ、送信が完了していない場合は CU_MAILS_SEND のビットが 1 となります。
送信がエラーとなった場合は、CU_MAILS_ERR のビットが 1 となり、下位 6 ビットのいずれかでエラーの種類を示します。

[6] Short と Integer のオーバーロードが定義されています。

CU_MailEnableReceive()

```
CU_STATUS CU_MailEnableReceive(CU_HANDLE hDev, long Mrb, BOOL flgEnable)

Function CU_MailEnableReceive(ByVal hDev As Long, ByVal Mrb As Long,
                              ByVal flgEnable As Long) As Long

Function CU_MailEnableReceive(ByVal hDev As System.IntPtr, ByVal Mrb As Integer,
                              ByVal flgEnable As Integer) As Integer
```

hDev : デバイスのハンドル
Mrb : メール受信の許可/禁止を設定する受信バッファ(MRB)を指定(0-1)
flgEnable : 許可または禁止を指定
0以外 : メール受信を許可
0 : メール受信を禁止

MR0CR、または、MR1CR の RDY ビットを操作することで、指定のメール受信バッファ(MRB)へのメール受信の許可/禁止を設定します。メール受信バッファはメールを受信すると、新たなメールの受信が自動的に禁止されますので、再度許可する場合にもこの関数を呼び出してください。
まだ取り出していない受信データがある場合、メール受信を許可することでデータが消去されますのでご注意ください。

CU_MailReceive()

```
CU_STATUS CU_MailReceive(CU_HANDLE hDev, long Mrb, void *pData, long nData, long *pSA, long *pnRcv)

Function CU_MailReceive(ByVal hDev As Long, ByVal Mrb As Long, ByRef pData As Any,
                        ByVal nData As Long, ByRef pSA As Long, ByRef pnRcv As Long) As Long

Function CU_MailReceive(ByVal hDev As System.IntPtr, ByVal Mrb As Integer,
                        ByVal pData() As Byte[7], ByVal nData As Integer,
                        ByRef pSA As Integer, ByRef pnRcv As Integer) As Integer
```

hDev : デバイスのハンドル
Mrb : メールデータを取り出す受信バッファ(MRB)の番号(0-1)
pData : [出力]受信データの格納先
nData : pData に格納可能なバイト数
pSA : [出力]メールの送信元ステーション
pnRcv : [出力]受信メールのサイズ

MKY43 のメール受信バッファ(MRB)から受信したメールデータを読み出します。
nData が受信メールのサイズよりも小さい場合は、格納可能なバイト数だけ pData に格納されます。CUnet のメールサイズの最大値は 256 バイトですので、nData が 256 以上であれば全てのメールを格納できます。
受信データが無い場合は、戻り値として CU_DATA_NOT_FOUND が返ります。

[7] Short と Integer のオーバーロードが定義されています。

CU_MailReadReceiveStatus()

```
CU_STATUS CU_MailReadReceiveStatus(CU_HANDLE hDev, DWORD *pStatus)

Function CU_MailReadReceiveStatus(ByVal hDev As Long, ByRef pStatus As Long) As Long

Function CU_MailReadReceiveStatus(ByVal hDev As System.IntPtr, ByRef pStatus As Integer) As Integer
```

hDev　　：デバイスのハンドル
pStatus：[出力]メール受信のステータス。以下のビットの組み合わせ
　　0x01 ： MRB0に受信データがある
　　0x02 ： MRB1に受信データがある

メール受信バッファ(MRB)のメール受信状態を返します。

□ 「MKY43」のその他の機能用関数

CU_Ping()

```
CU_STATUS CU_Ping(CU_HANDLE hDev, long SA)

Function CU_Ping(ByVal hDev As Long, ByVal SA As Long) As Long

Function CU_Ping (ByVal hDev As System.IntPtr, ByVal SA As Integer) As Integer
```

hDev : デバイスのハンドル
SA　: PING の送信先ステーション(0-63)

指定ステーションに PING を送信します。
クエリ実行中で PING が送信できない場合、戻り値として CU_TRYING_QUERY が返ります。
Ping は GMM モードでは利用できません。

CU_Query()

```
CU_STATUS CU_Query(CU_HANDLE hDev, long SA, long *pType, DWORD Timeout)

Function CU_Query(ByVal hDev As Long, ByVal SA As Long,
                  ByRef pType As Long, ByVal Timeout As Long) As Long

Function CU_Query(ByVal hDev As System.IntPtr, ByVal SA As Integer,
                  ByRef pType As Integer, ByVal Timeout As Integer) As Integer
```

hDev　: デバイスのハンドル
SA　: クエリを実行するステーション(0-63)
pType　: [出力]クエリの結果。以下の値のいずれか
　0x00 : MEM ステーション(フレームオプション 0)
　0x01 : MEM ステーション(フレームオプション 1)
　0x02 : IO ステーション(フレームオプション 0)
　0x03 : IO ステーション(フレームオプション 1)
　0x04 : MEM ステーションの拡張領域
Timeout : 応答の待ち時間(msec 単位)

指定ステーションに対するクエリを実行し、ステーションタイプを調べます。
待ち時間が経過した場合、戻り値として CU_QUERY_TIMEOUT が返ります。
クエリは GMM モードで実行可能です。

サポート情報

『CUstation-USB』、『CUstation-LAN』に関する情報、最新のファームウェア、ユーティリティなどは弊社ホームページにてご案内しております。また、お問い合わせ、ご質問などは下記までご連絡ください。

テクノウェーブ(株)
URL : http://www.techw.co.jp
E-mail : support@techw.co.jp

(1) 本書、および本製品のホームページに掲載されている応用回路、プログラム、使用方法などは、製品の代表的動作・応用例を説明するための参考資料です。これらに起因する第三者の権利(工業所有権を含む)侵害、損害に対し、弊社はいかなる責任も負いません。

(2) 本書の内容の一部または全部を無断転載することをお断りします。

(3) 本書の内容については、将来予告なしに変更することがあります。

(4) 本書の内容については、万全を期して作成いたしましたが、万一ご不審な点や誤り、記載もれなど、お気づきの点がございましたらご連絡ください。

改訂記録

年月	版	改訂内容
2009 年 8 月	初	
2009 年 11 月	2	・64bit 版ドライバに対応した記述に変更
2012 年 6 月	3	・Ver.1.3.0.x に対応
2017 年 6 月	4	・対応 OS を変更