アプリケーションプログラム

インターフェース仕様書

第１版　　2001 年　09 月 20 日

目　次

機能　　デバイスの認識

書式　　**int　getdevice(void);**

説明　　**接続装置を１つ、認識します**

引数　　**ありません**

出力　　**ありません**

注意　　**すべての操作に先立ち、必ず装置認識をおこなう必要があります**

戻値　　**装置認識に成功すると「OK_1394(0)」が戻り値となります**
その他のエラー情報については、関数戻り値一覧を参照ください

使用例

```
int         ret;

ret = getdevice( );                                    // デバイス認識
if( ret == 0 ){
      //成功時の処理
}
```

機能　　バスリセット実行

書式　　**int　busreset(void);**

説明　　バスリセットを実行します

引数　　ありません

出力　　ありません

戻値　　関数戻り値一覧表を参照

使用例

```
int  ret;

ret = busreset();            //バスリセット実行
if( ret == 0 ){
    // 正常時の処理
}
```

機能　　バスリセット待機

書式　　**int　busresnoty(void);**

説明　　バスリセット発生を待機します

引数　　ありません

出力　　ありません

戻値　　関数戻り値一覧を参照

注意　・この関数を呼び出すと、バスリセットが発生するまで内部で待機します
・復帰はバスリセットが発生するか、接続ノードがなくなったときです
・２重呼び出しはできません

使用例

```
int  ret;

ret = busresnoty();          //バスリセット待機
if( ret == 0 ){
    // 待機発生時の処理
}
```

機能　　ジェネレーションカウント値の確認

書式　　**int　get_generation(ULONG *dat);**

説明　　ジェネレーションカウント値を確認します

引数　　***dat**　　格納バッファ

出力　　***dat**　　取得したジェネレーションカウント値

戻値　　関数戻り値一覧を参照

使用例

```
int  ret;
ULONG  dat;

ret = get_generation( &dat);          //ジェネレーションカウント確認
if( ret == 0 ){
    // 正常時の処理
}
```

機能　　ノードアドレスの確認

書式　　**int　get_address(ULONG *dat);**

説明　　デバイスのバスおよびノード番号を確認します

引数　　*dat　　格納バッファ
　　dat[0]=0:　相手ノードの情報を取得します
　　dat[0]=1:　本機（パソコン）の情報を取得します

出力　　*dat　　取得したバス・ノード番号が格納されます
　　dat[0]:　ノード番号
　　dat[1]:　バス番号

戻値　　関数戻り値一覧を参照

使用例

```
int  ret;
ULONG  dat[2], nodeinfo, businfo;

dat[0] = 0;
ret = get_address( &dat[0] );        //相手情報取得
if( ret == 0 ){
    nodeinfo = dat[0];               //ﾉｰﾄﾞ番号
    businfo = dat[1];                //バス番号
}
```

機能　　バス転送速度の確認

書式　　**int　get_speedbetween(ULONG *dat);**

説明　　デバイスのもつ、バス転送速度を確認します

引数　　*dat　　格納バッファ
　　dat = 0:　　相手ノードの情報を取得します
　　dat = 1:　　本機（パソコン）の情報を取得します

出力　　*dat　　取得したバス速度定義値が格納されます
　　SPEED_S100　　１００Ｍｂｐｓ最大転送速度をサポート
　　SPEED_S200　　２００Ｍｂｐｓ最大転送速度をサポート
　　SPEED_S400　　４００Ｍｂｐｓ最大転送速度をサポート

戻値　　関数戻り値一覧を参照

使用例

```
int  ret;
ULONG  dat;

dat = 0;
ret = get_speedbetween(&dat);          //相手情報を確認
if( ret == 0 ){
    // 正常時の処理
}
```

機能　　ギャップカウント値の設定

書式　　**int　set_properties(ULONG *dat);**

説明　　ギャップカウント値を設定します

引数　　*dat　　ギャップカウント値を設定します（範囲：０～６３（１０進数））

出力　　ありません

戻値　　関数戻り値一覧を参照

使用例

```
int  ret;
ULONG  dat=0x3f;

ret = set_properties(&dat);          //ギャップカウント設定
if( ret == 0 ){
    // 正常時の処理
}
```

機能　　アシンクロナスリードリクエスト実行

書式　　**int　async_read(ASYNC *areq);**

説明　　アシンクロナスリードリクエスト処理を実行します

引数　　*areq　　アシンクロナスリクエスト情報構造体　　構造体内容は次のとおり

```
typedef  struct {
    USHORT  offset_h;       I 相手先アドレス　上位 16bit
    ULONG   offset_l;       I 相手先アドレス　下位 32bit
    ULONG   speed;          I 転送速度          （※１）
    ULONG   dlen;           I リクエストデータサイズ　（※２,※３）
    ULONG   *buf;           O リードデータ格納ポインタ
} ASYNC;
```

出力　　*buf　読み込んだデータが格納されます

戻値　　関数戻り値一覧を参照

記事　　※１　転送速度には、次の定数を定義します
　　SPEED_S100=100Mbps、 SPEED_S200=200Mbps，　SPEED_S400=400Mbps
※２　リクエストデータサイズはクワッドレット（４バイト）単位としてください
※３　１回に指定できる最大データサイズは、転送速度により次のようになります
　　SPEED_S100=512 バイト、SPEED_S200=1024 バイト、SPEED_S400=2048 バイト

使用例

```
int  ret;
ULONG  buf;
ASYNC  areq;

areq.offset_h = 0xffff;             //上位アドレス指示
areq.offset_l = 0xf0000404;         //下位アドレス指示
areq.speed = SPEED_S100;            //転送速度
areq.dlen = 4;                      //データサイズ
areq.buf = &buf;                    //バッファ指定
ret = async_read( &areq );          //リクエスト実行
if( ret == 0 ){
    //成功時の処理
}
```

機能　　アシンクロナスライトリクエスト実行

書式　　**int　async_write(ASYNC *areq);**

説明　　アシンクロナスライトリクエスト処理を実行します

引数　　***areq**　　アシンクロナスリクエスト情報構造体　　構造体内容は次のとおり

```
typedef  struct {
    USHORT  offset_h;        | 相手先アドレス　上位 16bit
    ULONG   offset_l;        | 相手先アドレス　下位 32bit
    ULONG   speed;           | 転送速度          （※１）
    ULONG   dlen;            | リクエストデータサイズ　（※２、※３）
    ULONG   *buf;            | ライトデータ格納ポインタ
} ASYNC;
```

出力　　ありません

戻値　　関数戻り値一覧を参照

記事　　※１　転送速度には、次の定数を定義します
　　SPEED_S100=100Mbps、 SPEED_S200=200Mbps，　SPEED_S400=400Mbps
※２　リクエストデータサイズはクワッドレット（４バイト）単位としてください
※３　１回に指定できる最大データサイズは、転送速度により次のようになります
　　SPEED_S100=512 バイト、SPEED_S200=1024 バイト、SPEED_S400=2048 バイト

使用例

```
int  ret;
ULONG  buf[128], dat;
ASYNC  areq;

buf[0] = 0x12345678;              //書き込みデータ
buf[1] = 0x87654321;
areq.offset_h = 0xffff;           //上位アドレス指示
areq.offset_l = 0x00000000;       //下位アドレス指示
areq.speed = SPEED_S100;          //転送速度
areq.dlen = 8;                    //データサイズ
areq.buf = &buf[0];               //バッファ指定
ret = async_write( &areq );       //リクエスト実行
if( ret == 0 ){
    //成功時の処理
}
```

機能　　アシンクロナスロックリクエスト実行

書式　　**int　async_lock(ASYNC *areq);**

説明　　アシンクロナスロックリクエスト処理を実行します

引数　　*areq　　アシンクロナスリクエスト情報構造体　　構造体内容は次のとおり

```
typedef  struct {
    USHORT  offset_h;       I 相手先アドレス　上位 16bit
    ULONG   offset_l;       I 相手先アドレス　下位 32bit
    ULONG   speed;            予約                    （※１）
    ULONG   dlen;           I リクエストデータサイズ　（※２）
    ULONG   *buf;           IO ロックデータポインタ　（※３）（※４）
} ASYNC;
```

出力　　*buf　応答データが格納されます

戻値　　関数戻り値一覧を参照

記事　　※１　実際の転送速度は 100Mbps に設定されます
※２　かならず「８」を設定します　(arg_value=1Quadret，data_value=1Quadret)
※３　バッファには、次の順に設定データを指定します
buf[0]　拡張トランザクションコード
設定できる拡張トランザクションコードは次のとおり
MASK_SWAP　または　COMPARE_SWAP　のいずれか
buf[1]　arg_value 値を設定します
buf[2]　data_value 値を設定します
※４　ロック処理結果データは、buf[0]に格納されます

注意　・arg_value=１要素、data_value=１要素の処理をおこないます
・転送速度は 100Mbps に固定されます

使用例

```
int  ret;
ULONG  dat[3], nowd;
ASYNC  areq;

areq.offset_h = 0xffff;             //上位アドレス指示
areq.offset_l = 0x00000000;         //下位アドレス指示
areq.dlen = 8;                      //固定値
dat[0] = COMPARE_SWAP;              //compare_swap
dat[1] = 0xffff;                    //arg 値指定
dat[2] = 0xfffe;                    //data 値指定
areq,buf = &dat;
ret = async_lock( &areq );          //リクエスト実行
if( ret == 0 ){
    nowd = dat[0];                  //成功時の処理
}
```

機能　　メモリマッピングの実行

書式　　**int　mem_map(MAP_MEM *mapmem);**

説明　　外部ノードからのアシンクロナスパケットに応答する､エリアマッピングをおこないます｡マッピングにより外部ノードからアシンクロナスリクエストがあった場合､データ処理をおこなうとともに、システムで自動応答します

引数　　***mapmem**　　メモリマッピング情報構造体　　構造体内容は次のとおり

```
typedef  struct {
    USHORT  index;          | マップインデックス番号（※１）
    USHORT  offset_h;       | アドレス　上位 16bit
    ULONG   offset_l;       | アドレス　下位 32bit （※２）
    ULONG   dlen;           | マップサイズ（※３）
    ULONG   accesstype;     | マップに対するアクセスタイプ（※４）
    ULONG   options;        | アクセス時の通知情報を設定します（※５）
    ULONG   *buf;           | データ格納ポインタ
} MAP_MEM;
```

戻値　　関数戻り値一覧を参照

記事　　※１　マップエリアの番号を指定します
　　０～３で指定する４つのマップエリア（インデックス）を管理します
※２　４バイト境界からマッピングします（下位２bit は０指定です）
※３　マップサイズは、クワッドレット（４バイト）単位としてください
　　また１エリアでマップできる最大サイズは、６４０００バイトです
※４　このマップエリアに対し相手ノードからリクエストが発生した場合､どのようなリクエストの時にデータ処理し、レスポンスを返答するか指定します
　　指定値は次のとおりで､「ＯＲ条件」で値を指定します
　　FLAG_READ :　リードリクエストで、データ処理後レスポンスを送信します
　　FLAG_WRITE:　ライトリクエストで、データ処理後レスポンスを送信します
　　FLAG_LOCK :　ロックリクエストで、データ処理後レスポンスを送信します
※５　マップエリアに対し、どのようなリクエストが発生したらアプリケーションに通知するかを指定します
　　この関数は「指定」をおこなうだけで、実際にアプリケーションにリクエスト発生を通知するためには、別関数である「マッピングエリアアクセス待機関数」を使用します。この関数での指定値は次のとおりで､「ＯＲ条件」で値を指定します

0 :　　リクエスト発生を通知しません
FLAG_READ:　リードリクエスト発生時､｢マッピングエリアアクセス待機関数｣の指示があれば､リクエスト発生を通知できるように設定します
FLAG_WRITE: ライトリクエスト発生時､｢マッピングエリアアクセス待機関数｣の指示があれば､リクエスト発生を通知できるように設定します
FLAG_LOCK:　ロックリクエスト発生時､｢マッピングエリアアクセス待機関数｣の指示があれば､リクエスト発生を通知できるように設定します

注意　・次のアドレス空間はマッピングできません
　　0xffff_f0000000　～　0xffff_f00007ff　の空間
・マップエリアにおけるマッピングアドレスのオーバーラップなどの管理はしません
　（インデックス番号が違っても、マップアドレスを同じにするなど）
　またオーバラップアドレス指定時の動作については保証されません
・通知情報を設定するばあい、必ずアクセスタイプで設定した情報をセットアップしてください
・｢accesstype｣で指定しない｢options｣を設定することはできません
・１度マッピングをおこなうと、その要素を「メモリ・アンマッピング」で解放するまでは、同じインデックス番号でのマッピング実行はできません
・マッピングしたメモリは、使用終了時に必ず「メモリ・アンマッピング」処理により解放してください。リソースが残ったままとなります
・ケーブル抜けなどで接続デバイスがなくなると、すべてのマッピングエリアは自動解放されます

使用例

```
MAP_MEM  mapmem;
ULONG  buf[512];
int  ret;

memset( &buf[0], 0, sizeof(buf) );
mapmem.index = 0;                              // マップインデックス番号
mapmem.offset_h = 0xffff;                      // 上位アドレス
mapmem.offset_l = 0;                           // 下位アドレス
mapmem.dlen = sizeof(buf);                     // マップサイズ
mapmem.buf = &buf[0];                          // マップデータ
mapmem.accesstype = FLAG_READ | FLAG_WRITE;    // アクセスタイプ
mapmem.options = = FLAG_READ | FLAG_WRITE;     // アプリケーション通知タイプ
ret = mem_map( &mapmem );                      // マッピング処理
if( ret == 0 ){
    //成功時の処理
}
```

機能　　マッピングメモリ・ライト

書式　　**int　write_map(MAP_MEM *areq);**

説明　　マップしたメモリ空間にデータを上書きします。マップエリアの一部データを書き込むことも可能です

引数　　***mapmem**　　メモリマッピング情報構造体　　構造体内容は次のとおり

```
typedef  struct {
    USHORT  index;          I マップインデックス番号（※１）
    USHORT  offset_h;       I 書き込みアドレス　先頭上位 16bit
    ULONG   offset_l;       I 書き込みアドレス　先頭下位 32bit　(※２)
    ULONG   dlen;           I 書き込みサイズ（※３）
    ULONG   accesstype;     未使用
    ULONG   options;        未使用
    ULONG   *buf;           I データ格納ポインタ
} MAP_MEM;
```

戻値　　関数戻り値一覧を参照

補足

※１　マップをおこなった、マップエリア番号（０～３）を指定します
※２　４バイトバウンダリのアドレス値を指定します
※３　書き込みデータサイズは、クワッドレット（４バイト）単位としてください

注意　・事前にメモリマッピングをおこなう必要があります
・マッピングしていない空間に書き込みをおこなったばあい、エラーとなります
・マッピングアドレス範囲外への書き込みは無視されます

使用例

```
int  ret;
ULONG  buf[512], i;
CHAR  *p;
MAP_MEM  areq;

p = (CHAR *)&buf[0];                //書き込みデータ設定
for( i=0; i<sizeof(buf); i++ ){
    *p++ = i;
}
areq.index = 0;                     //マップインデックス番号
areq.offset_h = 0xffff;             //先頭アドレス上位
areq.offset_l = 0;                  //先頭アドレス下位
areq.dlen = sizeof(buf);            //サイズ
areq.buf = &cbuf[0];                //バッファ
ret = write_map( &areq );
if( ret == 0 ){
    //正常時の処理
}
```

機能　　マッピングメモリ・リード

書式　　**int　read_map(MAP_MEM *areq);**

説明　　マップしたメモリ空間データを読み出します。マップエリアの一部データを読み出すことも可能で読み出されたデータは指定バッファ先頭から格納されます

引数　　*mapmem　　メモリマッピング情報構造体　　構造体内容は次のとおり

```
typedef  struct {
    USHORT  index;          I マップインデックス番号（※１）
    USHORT  offset_h;       I 読み出しアドレス　先頭上位 16bit
    ULONG   offset_l;       I 読み出しアドレス　先頭下位 32bit （※２）
    ULONG   dlen;           I 読み出しサイズ（※３）
    ULONG   accesstype;     未使用
    ULONG   options;        未使用
    ULONG   *buf;           O データ格納ポインタ
} MAP_MEM;
```

戻値　　関数戻り値一覧を参照

補足

※１　マップをおこなった、マップエリア番号（０～３）を指定します
※２　４バイトバウンダリのアドレス値を指定します
※３　読み出しデータサイズは、クワッドレット（４バイト）単位としてください

注意　・事前にメモリマッピングをおこなう必要があります
・マッピングしていないアドレスデータを読み出そうとしたばあい、エラーとなります
・マッピングアドレス範囲外からの読み出しは無視されます

使用例

```
int  ret;
ULONG  buf[512];
MAP_MEM  areq;
areq.index = 0;                  //マップインデックス番号
areq.offset_h = 0xffff;          //読み出し先頭アドレス上位
areq.offset_l = 0;               //読み出し先頭アドレス下位
areq.dlen = sizeof(buf);         //読み出しサイズ
areq.buf = &buf[0];              //読み出しバッファ
ret = read_map( &areq );
if( ret == 0 ){
    //正常時の処理
```

機能　　マップメモリ・リクエスト待機

書式　　**int　wait_map(ULONG *dat);**

説明　　マップエリアに対して外部ﾉｰﾄﾞからのリクエスト発生を待機し､指定リクエストがあれば最新のリクエスト内容を通知します

引数　　*dat　　待機する、リクエスト要因
　　dat[0]　　マップインデックス番号（０～３）　（※１）

出力　　*dat　　発生したリクエスト要因（下記いずれかになります）
　　dat[0] = FLAG_READ　　このマップエリアに、リードリクエストが発生
　　FLAG_WRITE　　このマップエリアに、ライトリクエストが発生
　　FLAG_LOCK　　このマップエリアに、ロックリクエストが発生
　　その他　　エラー

戻値　　関数戻り値一覧を参照

記事　　※１　マッピングをおこなっている、マップ番号を指定します
　　指定したマップエリアへのアクセスがあった場合に、待機が完了しリクエスト要因を通知します

注意　・待機をおこなうためには、あらかじめ「メモリ・マッピング」処理によりメモリをマップし、かつ「メモリ・マッピング」設定項目「options」で、あらかじめ通知情報を設定しておく必要があります
・この関数を呼び出すと、指定リクエスト発生まで内部で待機します
・アプリケーション終了時などでは、待機を必ずキャンセルしてください
（waitcancel_map 関数または mem_unmap 関数により、待機がキャンセルされます）

使用例

```
int  ret;
ULONG  dat;

/*  あらかじめ mem_map()にてマップしておきます  */
dat = 0;                              //マップ０指示
ret = wait_map( &dat );               //ﾘｸｴｽﾄ待機の実行
if( ret == 0 ){                       //要求ﾘｸｴｽﾄあれば復帰
    if( dat == FLAG_READ ){
        -------
    }
}
```

機能　　マップメモリ・リクエスト待機のキャンセル

書式　　**int　waitcancel_map(ULONG *dat);**

説明　　マップエリアに対するリクエスト待機をキャンセルします

引数　　*dat　　待機をキャンセルする、マップ番号（０～３）

出力　　ありません

戻値　　関数戻り値一覧を参照

注意　・強制的にリクエスト待機を解除するために本関数を使用します
　　・この関数を実行することでリクエスト待ちが解除され、エラーリターンします

使用例

```
int  ret;
ULONG  dat;

dat = 0;                              //マップ０指示
ret = waitcancel_map( &dat );         //待機のｷｬﾝｾﾙ
if( ret == 0 ){                       //本関数終了とともに待機ｷｬﾝｾﾙ
    //成功時の処理
}
```

機能　　メモリ・アンマッピング

書式　　**int　mem_unmap(ULONG *dat);**

説明　　マップしたエリアを解放します
アンマッピング以後、このエリアへの外部ノードからのリクエストではシステムがエラー返却します

引数　　***dat**　　マッピングした、マップインデックス番号を指示します
０～３の指定が有効です

出力　　ありません

戻値　　関数戻り値一覧を参照

注意　・マッピングしたメモリは、アプリケーション終了時などには必ず本関数を使用して解放してください

使用例

```
int  ret;
ULONG  dat=0;                    //マップ０指示

ret = mem_unmap( &dat );         //ｱﾝﾏｯﾌﾟの実行
if( ret == 0 ){
    //正常時の処理
}
```

機能　　アイソクロナス・バンド幅指定

書式　　**int　iso_alloc_band(ULONG *dat);**

説明　　アイソクロナス転送における、バンド幅（帯域）を指定します

引数　　*dat　　指定情報
　　dat[0]:　　転送速度　　※１
　　dat[1]:　　転送１パケットサイズ長（バイト）　※２

出力　　ありません

戻値　　関数戻り値一覧を参照

記事　　※１　転送速度設定は次のとおり
　　SPEED_S100　　100Mbps での転送指定
　　SPEED_S200　　200Mbps での転送指定
　　SPEED_S400　　400Mbps での転送指定
　　※２　必ず、Quadlet 単位で設定します
　　また指定する転送速度により上限値が次のように制限されます
　　SPEED_S100 の場合　　1024 バイト設定上限
　　SPEED_S200 の場合　　2048 バイト設定上限
　　SPEED_S400 の場合　　4096 バイト設定上限

注意　・１度指定をおこなうと、「アイソクロナス・バンド幅解放」処理で解放するまで、再指定はできません
・指定したバンド幅は、アプリケーション終了時に必ず「アイソクロナス・バンド解放」処理により解放してください
・バスリセットが発生した場合、あるいは接続ノードがなくなると自動解放されます

使用例

```
int  ret;
ULONG  dat[2];

dat[0] = SPEED_S100;
dat[1] = 512;
ret = iso_alloc_band( &dat[0] );          //バンド幅指定実行
if( ret == 0 ){
    // 正常時の処理
}
```

機能　　アイソクロナス・バンド幅解放

書式　　**int　iso_free_band(void);**

説明　　指定した、アイソクロナス・バンド幅（帯域）を解放します

引数　　ありません

出力　　ありません

戻値　　関数戻り値一覧を参照

注意　・「アイソクロナス・バンド幅指定」によるバンド幅は、処理終了時には必ず本関数で解放してください

使用例

```
int  ret;

ret = iso_free_band();
if( ret == 0 ){
    // 正常時の処理
}
```

機能　　アイソクロナス・チャンネル指定

書式　　**int　iso_alloc_channel(ULONG *dat);**

説明　　アイソクロナス転送に必要となる、転送チャンネルをバスに指定します

引数　　*dat　　要求するチャンネル番号　（０～６３）

出力　　ありません

戻値　　関数戻り値一覧を参照

注意　・１度指定をおこなうと、「アイソクロナス・チャンネル解放」処理で解放するまでは再指定できません
・指定したチャンネルは、アプリケーション終了時に必ず「アイソクロナス・チャンネル解放」処理により解放してください
・バスリセットが発生した場合、あるいは接続ノードがなくなると自動解放されます

使用例

```
int  ret;
ULONG  dat;

dat= 0;                                    // 要求するチャンネル
ret = iso_alloc_channel( &dat );
if( ret == 0 ){
    //成功時の処理
}
```

機能　　アイソクロナス・チャンネル解放

書式　　**int　iso_free_channel(void);**

説明　　指定した、アイソクロナス・チャンネルを解放します

引数　　ありません

出力　　ありません

戻値　　関数戻り値一覧を参照

注意　・「アイソクロナス・チャンネル指定」で指定したチャンネルは、必ず本関数で解放してください

使用例

```
int  ret;

ret = iso_free_channel();
if( ret == 0 ){
    // 正常時の処理
}
```

機能　　アイソクロナス・転送資源設定

書式　　**int　iso_set_res (ISO_RES *req);**

説明　　アイソクロナス送受信をおこなうために、転送資源を設定します

引数　　*req　　アイソクロナス資源情報構造体　　構造体内容は次のとおり

```
typedef  struct {
    ULONG  code;          I 送受信設定コード（※１）
    ULONG  ch;            I 転送ｃｈ        （※２）
    ULONG  speed;         I 転送速度        （※３）
    ULONG  pktsize;       I １パケットサイズ（※４）
    ULONG  bufsize;       I バッファサイズ  （※５）
} ISO_RES;
```

出力　　ありません

戻値　　関数戻り値一覧を参照

記事　　※１　送受信設定コードを指示します
アイソクロナス送信資源設定では「ISO_WRITE」を、アイソクロナス受信資源設定では「ISO_READ」を、いずれか設定します
※２　ｃｈには０～６３のいずれかを指定します
※３　次のいずれかを指定します
SPEED_S100=100Mbps，SPEED_S200=200Mbps，SPEED_S400=400Mbps
※４　パケットサイズには、次の制約があります
・Quadret 単位であること
・転送速度により、次の最大値があります
SPEED_S100 の場合　1024 バイト上限
SPEED_S200 の場合　2048 バイト上限
SPEED_S400 の場合　4096 バイト上限
※５　バッファサイズには、次の制約があります
・最大サイズ＝64000 バイト
・パケットサイズの正数倍であること

注意　・１度にセットアップできる資源は１つだけです。そのため送信と受信を同時におこなうことはできません
・アプリケーション終了時には、「アイソクロナス・転送資源解放」処理で資源解放をしてください
・接続ノードがなくなると、自動解放されます

使用例

```
int  ret;
ISO_RES  req;

req.code = ISO_WRITE;                    // 送信資源を設定
req.ch = 0;                              // ch 設定
req.speed = SPEED_S100;                  // 速度設定
req.pktsize = 512;                       // パケットサイズ
req.bufsize = req.pktsize * 20;          // バッファサイズ
ret = iso_set_res( &req );
if( ret == 0 ){
    //正常時の処理
}
```

機能　　アイソクロナス・転送資源解放

書式　　**int　iso_free_res (void);**

説明　　割り当てた、転送資源を解放します

引数　　ありません

出力　　ありません

戻値　　関数戻り値一覧を参照

使用例

```
int  ret;

ret = iso_free_res();
if( ret == 0 ){
    //成功時の処理
}
```

機能　　アイソクロナス・転送情報設定

書式　　**int　iso_set_info(ISO_INFO *req)**

説明　　アイソクロナス送受信をおこなうために、転送情報を設定します

引数　　*req　　アイソクロナス情報設定構造体　　構造体内容は次のとおり

```
typedef  struct {       // ｱｲｿｸﾛﾅｽｱﾀｯﾁ
    ULONG  code;            ｜ 送受信設定コード（※１）
    ULONG  pktsize;         ｜ １パケットサイズ（※２）
    ULONG  bufsize;         ｜ バッファサイズ　（※３）
    ULONG  synccode;        ｜ SYNC_CODE　　（※４）
    ULONG  tagcode;         ｜ TAG_CODE　　（※５）
    ULONG  syncflg;         ｜ SYNC ﾌﾗｸﾞ　　（※６）
} ISO_INFO;
```

出力　　ありません

戻値　　関数戻り値一覧を参照

記事　　※１　送受信設定コードを指示します
　　送信の場合は「ISO_WRITE」を、受信の場合は「ISO_READ」を設定します
※２　「アイソクロナス・資源設定」で指定したパケットサイズを指定します
※３　「アイソクロナス・資源設定」で指定したバッファサイズを指定します
※４　SYNC コード　０～１５のいずれかを指定します
※５　TAG コード　０から３のいずれかを指定します
※６　同期化コードを指定します
　　SYNCH_SY=SYNC コードに同期
　　SYNC_TAG=TAG コードに同期

注意　・本関数実行前に、「アイソクロナス・転送資源設定」処理で資源設定をする必要があります
・アプリケーション終了時には、「アイソクロナス・転送情報解放」処理で設定情報を解放してください
・接続ノードがなくなると、設定情報は自動解放されます

使用例

```
int  ret;
ISO_INFO  req;

req.code = ISO_WRITE;
req.pktsize = 512;                    //１パケットサイズ
req.bufsize = req.pktsize * 20;       //バッファサイズ
req.synccode = 0;                     //SYNC_CODE
req.tagcode = 1;                      //TAG_CODE
req.syncflg = SYNC_SY;                //SYNC ﾌﾗｸﾞ
ret = iso_set_info( &req );
if( ret == 0 ){
    //正常時の処理
}
```

機能　　アイソクロナス・転送情報解放

書式　　**int　iso_free_info(void)**

説明　　アイソクロナス送受信の転送情報を解放します

引数　　ありません

戻値　　関数戻り値一覧を参照

使用例

```
int  ret;

ret = iso_free_info();
if( ret == 0 ){
    //成功時の処理
}
```

機能　　アイソクロナス・バッファセット

書式　　**int　iso_set_buf(ULONG *dat);**

説明　　アイソクロナス送受信用、データバッファ設定をおこないます

引数　　*dat　　バッファアタッチ情報

dat[0]:	I 送受信設定コード	（※１）
dat[1]:	I 管理インデックス番号	（※２）
dat[2];	I ﾊﾞｯﾌｧｻｲｽﾞ	（※３）
dat[3]:	I/O　データバッファポインタ	（※４）

戻値　　関数戻り値一覧を参照

記事　　※１　送受信設定コードを指示します
送信の場合は「ISO_WRITE」を、受信の場合は「ISO_READ」を設定します
※２　０～３を指定してください
※３　「アイソクロナス送受信・転送資源設定」で設定したバッファサイズを指定します
※４　送受信をおこなうデータバッファポインタを指示します

注意　・この関数を呼び出すと、処理終了までバッファは内部でロックされます
・送受信が終了するかキャンセルされるまで、関数はロックされます
・途中キャンセルする場合、「アイソクロナス・バッファ解放」処理を実行します
・データバッファサイズは「アイソクロナス・転送資源設定」処理で指定したバッファサイズになります。そのためサイズ割り当てには注意してください
・終了通知があるまで、同じインデックス番号でのバッファセットはできません
・割り当てるバッファについては、バッファフルになるように割り当ててください
送信の場合、割り当てられたバッファフルになるように送信をおこない、関数終了します
受信の場合、バッファフルになるように受信をおこない、関数終了します
・アプリケーション終了時には、「アイソクロナス・バッファ解放」処理によりバッファを解放してください
・接続ノードがすべてなくなると、すべてのバッファは自動解放されます

使用例

```
int  ret;
ULONG  dat[4];
ULONG  dbuf[2048];

dat[0] = ISO_WRITE;                          //送信設定
dat[1] = 0;                                  //ｲﾝﾃﾞｯｸｽ番号
dat[2] = sizeof(dbuf);                       //ﾊﾞｯﾌｧｻｲｽﾞを指定
for( i=0; i<sizeof(dbuf)/4; i++ ){
    dbuf[i] = i;
}
dat[3] = &dbuf[0];

ret = iso_set_buf( &dat[0] );                //処理実行
if( ret == 0 ){                              //送信完了かエラーで復帰
    //正常時の処理
}
```

機能　　アイソクロナス・バッファ解放

書式　　**int　iso_free_buf(void)**

説明　　アイソクロナス・バッファセットによりシステムに保留されたバッファ要素をすべて解放します

引数　　ありません

戻値　　関数戻り値一覧を参照

注意　・本関数により、内部で保留されたバッファ要素をすべて解放します

使用例

```
    int  ret;

   /* 事前にバッファセットがおこなわれています */
    ret = iso_free_buf();          //処理実行
    if( ret == 0 ){                //このとき保留されたバッファ要素が解放されます
        //正常時の処理
    }
```

機能　　アイソクロナス・送受信開始

書式　　**int　iso_start(ULONG *dat);**

説明　　アイソクロナス送受信指示をおこないます

引数　　*dat　　送受信設定コードを指示します
送信実行の場合　　ISO_WRITE 設定
受信実行の場合　　ISO_READ 設定

出力　　ありません

戻値　　関数戻り値一覧を参照

使用例

```
    int  ret;
    ULONG dat;

   dat = ISO_WRITE;                          //送信を開始
    ret = iso_start(&dat);                   //処理実行
    if( ret == 0 ){
        //正常時の処理
    }
```

機能　　アイソクロナス・送受信のストップ

書式　　**int　iso_stop(void);**

説明　　アイソクロナス送受信を終了します

引数　　ありません

出力　　ありません

戻値　　関数戻り値一覧を参照

使用例

```
int  ret;

ret = iso_stop();              //処理実行
if( ret == 0 ){
    // 正常時の処理
}
```

機能　　アイソクロナス・サイクルタイム確認

書式　　**int　iso_query_cycletime (ULONG *dat);**

説明　　現在のサイクルタイムを確認します

引数　　*dat　　指定情報

出力　　*dat　　獲得情報
　　dat[0]:　　クロック数
　　dat[1]:　　サイクル数
　　dat[2]:　　秒数（０～１２８）

戻値　　関数戻り値一覧を参照

使用例

```
int  ret;
ULONG  dat[3], clock, cycle, second;

ret = iso_ query_cycletime( &dat[0] );        //処理実行
if( ret == 0 ){
    clock = dat[0];
    cycle = dat[1];
    second = dat[2];
}
```

機能　　アイソクロナス・リソース確認

書式　　**int　iso_query_resources (ULONG *dat);**

説明　　速度に応じた、利用可能な帯域幅／チャンネルを確認します

引数　　*dat　　バス転送速度を指定します
　　dat[0]:　　SPEED_S100，SPEED_S200，SPEED_S400 のいずれかを指定します

出力　　*dat　　獲得情報
　　dat[0]:　　指定したバス速度を返します
　　dat[1]:　　残りの利用可能なバンド幅を返却します
　　dat[2]:　　残りの、利用可能なチャンネル（32～63）を返却します
　　dat[3]:　　残りの、利用可能なチャンネル（0～31）を返却します

戻値　　関数戻り値一覧を参照

使用例

```
int  ret;
ULONG  dat[4], band, ch_Hi, ch_Lo;

dat[0] = SPEED_S100;                          //100Mbps
ret = iso_ query_resources (&dat[0]);         //処理実行
if( ret == 0 ){
    band = dat[1];
    ch_Hi = dat[2];
    ch_Lo = dat[3];
}
```

関数戻り値　一覧

定義	説明
OK_1394	正常終了 ・要求された処理は、正常に終了した （本戻り値は、「０」にても判定可能）
PARAMETER_ERROR	要求したパラメータにエラーがある
HANDLE_ERROR	指定したデバイス名称では、デバイス処理できない
ACCESS_ERROR	ドライバ内部で、処理に失敗した
INTERNAL_ERROR	バッファを確保できないなどの、内部エラー
BUSY_ERROR	指示された要求はすでに実行中にある あるいは 指示された要求にたいしてのセットアップがおこなわれていない

定数　一覧

・アシンクロナス転送速度

定義	値	説明
SPEED_S100	1	100Mbps
SPEED_S200	2	200Mbps
SPEED_S400	3	400Mbps

・アシンクロナスフラグ

定義	値	説明
FLAG_READ	1	リード処理属性
FLAG_WRITE	2	ライト処理属性
FLAG_LOCK	4	ロック処理属性

・ロックトランザクション拡張コード

定義	値	説明
MASK_SWAP	1	MASK_SWAP 処理
COMPARE_SWAP	2	COMPARE_SWAP 処理

・アイソクロナス同期化コード

定義	値	説明
SYNC_SY	1	SYNC 同期
SYNC_TAG	2	TAG 同期

・アイソクロナス設定コード

定義	値	説明
ISO_READ	1	受信設定
ISO_WRITE	2	送信設定

構造体　一覧

・アシンクロナスリクエスト

```
typedef  struct _ASYNC {
    USHORT  offset_h;    Ｉ  先頭アドレス上位１６ビット
    ULONG   offset_l;    Ｉ  先頭アドレス下位３２ビット
    ULONG   speed;       Ｉ  転送速度
    ULONG   dlen;        Ｉ  処理データサイズ（バイト）
    ULONG   *buf;        ＩＯ  データバッファ
} ASYNC, *PASYNC;
```

・メモリマッピング

```
typedef  struct _MAP_MEM{
    USHORT  index;          Ｉ  マップインデックス番号
    USHORT  offset_h;       Ｉ  先頭アドレス　上位 16bit
    ULONG   offset_l;       Ｉ  先頭アドレス　下位 32bit
    ULONG   dlen;           Ｉ  マップサイズ
    ULONG   accesstype;     Ｉ  マップに対するアクセスタイプ
    ULONG   options;        Ｉ  アクセス時の通知情報を設定
    ULONG   *buf;           ＩＯ  データバッファ
} MAP_MEM, *PMAP_MEM;
```

・アイソクロナス資源設定

```
typedef  struct _ISO_RES {
    ULONG  code;             Ｉ  資源設定内容
    ULONG  ch;               Ｉ  転送ｃｈ
    ULONG  speed;            Ｉ  転送速度
    ULONG  pktsize;          Ｉ  １パケットサイズ
    ULONG  bufsize;          Ｉ  バッファサイズ
} ISO_RES, *PISO_RES;
```

・アイソクロナス情報設定

```
typedef  struct _ISO_INFO {
    ULONG  code;             I 情報設定内容
    ULONG  pktsize;          I パケットサイズ
    ULONG  bufsize;          I バッファサイズ
    ULONG  synccode;         I SYNC コード
    ULONG  tagcode;          I TAG コード
    ULONG  syncflg;          I SYNC フラグ
} ISO_INFO, *PISO_INFO;
```