HIToolbar Programming Guide

（1）HIToolbarのコンセプト

このドキュメントでは、HIToolbarについて解説します。このドキュメントを読む前に、Carbon EventのハンドリングやHIObjectなどの技術についても理解を深めておかれることをお勧めします。こうした技術については、別ドキュメントの「Carbon Event Manager Programming Guide」や「HIView Programming Guide」を参照してみてください。

・Toolbarとは何か？

・ウィンドウのタイトルバーの下に表示されるユーザインターフェースユニットです。
・プッシュボタンなどのコントロールを含む複数のToolbarアイテムを配置可能です。
・ユーザがToolbarアイテムをクリックすることで実行される各種機能を実装できます。
・Toolbarアイテムとしてはアイコンの他にポップアップメニューなども選択できます。
・ウィンドウタイトルバー右のToolbarボタンにより表示・非表示を切り替えられます。
・このボタンのCommand+クリックで表示状態（テキストのみ等）も変更できます。

ユーザは、幾つかの方法でToolbarアイテムの種類や並びをカスタマイズすることが可能です。まず、Command+マウスドラッグでアイテムの位置を移動させることができます（Mac OS X 10.2.4以降）。Toolbarがアイテムのドラッグに対応していれば、ダイレクトにアイテムの内容を差し替えることも可能です。また、そのアプリケーションにアイテム構成を変更するための「Customize Toolbar」シートを表示できる機能があれば、ToolbarボタンをCommand+Option+クリックすることで、それを表示させることが可能です。アイテムの表示状態やToolbar自身の表示モードなどは、アプリケーションの終了時点で、ユーザの Preferencesフォルダに保存することで、次回起動で再び有効にさせることが可能です。

・HIObjectとしてのToolbar

Carbon環境においては、HIToolbarはHIObjectのサブクラスです（Mac OS X 10.2以降）。HIToolbarはウィンドウとは独立しています。ToolbarはView上に描画されますが、それ自身がウィンドウのルートViewに埋め込まれる（embeded）形態を取ります。

（a）HIToolbarのモデル・ビュー・コントロール

Toolbarには、Toolbarアイテムにのリストデータが属しており、各アイテムは、その表示状態や動作を定義したプロパティを持ちます。ToolbarもToolbarアイテムも、その表示方式はHIViewモデルに準拠しており、それぞれ個々に用意されたView上に表示されることになります。ユーザがToolbarアイテムをクリックした場合には、Carbon Event Handler側でそのコマンドID に対応した処理が実行されます。ただし、もしそこに処理が記述されていない場合には、Standard Event Handlerに用意されている処理が実行されます。

（b）デリゲート（委任処理）

Toolbarを作成したりアイテムを加えたりする処理については、Carbon Eventのクラスとしてk EventClassToolbarが用意されています。Toolbarクラスに属する各種イベントを受け取れば、オリジナル処理を差し替えたり、別処理を追加したりすることも可能です。Toolbarクラス（kEventClassToolbar）でどのような種類のCarbon Eventを受け取ることが可能かについては、HIToolbar.hを参照してください。

また、HIToolbarSetDelegate()を用いると、イベントターゲットを別オブジェクトに差し替えることも可能です。この時、Carbon EventにはkEventTargetDontPropagate オプションが設定されていますので、イベントはそのターゲットより下層へ送られることはありません。

（c）Toolbarアイテム

HIToolbarItemオブジェクトはHIObjectのサブクラスです。それはHIToolbarオブジェクトに関連付けされていますが、そこに埋め込まれているわけではありません。Toolbarアイテムは「識別子」（Identifier）により認識することができ、数多くのスタンダードアイテムが用意されています。一般的なアイテムはアイコンボタン機能を有しますが、HIObjectのサブクラスとして独自機能を持たせたアイテムを作成することも可能です。Toolbarアイテムは、それ自身に関連する以下のようなデータ構造を持ちます。

・どのアイコンを表示するか決定するアイコンリファレンス（Icon Reference）
・アイコンの下に表示させる機能内容を記載したラベル（テキスト）
・そのアイテムの機能を解説するために表示させるヘルプタグ（Help Tag）
・Carbon Event Handlerで処理を分岐させるためのコマンドID（Command ID）

・ビルトイン機能

Toolbarには以下に示すようなビルトイン機能があり、複雑な追加処理が必要無いように考慮されています。

・アイテム構成を変更するためのシートウィンドウの表示。
・現在のToolbarの設定をユーザのプリファレンスに保存する。
・Toolbarの表示・非表示の切り替え。
・Toolbarアイテムのドラッグによる追加と並び替え。
・Toolbar上へのコンテキストメニューの表示。
・Toolbarの幅が狭い場合に非表示アイテムを自動でメニューアイテムにまとめる。

・システム要求

Toolbarは、Mac OS X 10.2以降で利可能です。Toolbarを埋め込むウィンドウはコンポジットモードである必要はありません。Toolbarに関連するAPIのMac OS X対応条件は、HIToolbar.hを参照してください。

（2）Toolbarのタスク

・Toolbarはどのように働くのか

ウィンドウにToolbarを表示させる手順は以下のとおりです。この手順においての状態変化については、すべてCarbon Eventとして（必要なら）受け取ることが可能です。

１．HIToolbarオブジェクトを作成した後、表示するまでは何も起こらない。
２．Toolbar表示時に対応するViewが作成されWindowのルートViewに埋め込まれる。
３．前回保存されたToolbarのアイテム設定状況が保存されているかチェックする。
４．前回のアイテム設定状況が見つかれば有効にし、なければディフォルト設定を使う。
５．アイテム設定状況から各アイテムのViewを作成しToolbar Viewに埋め込む。
６．オープンされているウィンドウの数だけ2、4、5の処理を繰り返す。
７．ユーザによるToolbarの設定変更はすぐに別ウィンドウへ通知される。
８．ユーザによるアイテム構成を変更用のシートのオープンはHandlerで処理する。

・Toolbarの作成

Toolbarの作成にはHIToolbarCreate()を用います。残念ならがInterface Builder経由で作成することはできません（現時点）。

```
OSStatus          err = noErr;
HIToolbarRef    toolbar;       // HIObjectRefと同等

err = HIToolbarCreate( CFSTR( "com.mycompany.carbontoolbar" ),
                kHIToolbarAutoSavesConfig ¦ kHIToolbarIsConfigurable, &toolbar );
               // 識別子とアトリビュートを渡してToolbarを作成する
               // kHIToolbarAutoSavesConfig・・・設定をプリファレンスとして保存
               // kHIToolbarIsConfigurable・・・ユーザが設定を変更可能とする
               // アトリビュートが必要ない場合にはkHIToolbarNoAttributesを渡す

InstallEventHandler( HIObjectGetEventTarget( (HIToolbarRef)toolbar ),
                   ToolbarDelegate, GetEventTypeCount( kToolbarEvents ),
                   kToolbarEvents, toolbar, NULL );
               // Carbon Event Handlerをインストールする
               // イベントターゲットを別に設定する時はHIToolbarSetDelegate()を使う

SetWindowToolbar( window, toolbar );  // ウィンドウにToolbarを設定する
ShowHideWindowToolbar( window, true, false );
               // Toolbarをアニメーションなしで表示（MacWindow.h参照）
CFRelease( toolbar );  // toolbarをリリースする（ウィンドウがリテインしている）
ChangeWindowAttributes( window, kWindowToolbarButtonAttribute, 0 );
               // タイトルバーにToolbarボタンを表示（MacWindow.h参照）
SetAutomaticControlDragTrackingEnabledForWindow( window, true );
               // コントロールの自動ドラッグトラッキングをONにする（Control.h参照）
```

・デリゲートを用いたイベントハンドリング

ユーザがToolbarを操作することで発生したCarbon Eventは、ディフォルトでToolbarへと送られて来ますが、HIToolbarSetDelegate()によってイベントターゲットを別オブジェクトに変更することも可能です。

以下のサンプルでは、4種類のCarbon Eventを処理するためのCarbon Event Handlerが作成されています。

```
static const EventTypeSpec kToolbarEvents[] =
```

```
{
    { kEventClassToolbar, kEventToolbarGetDefaultIdentifiers },
    // Toolbarにディフォルトアイテムの表示を要求された場合

    { kEventClassToolbar, kEventToolbarGetAllowedIdentifiers },
    // 構成変更用シートのアイテムの表示を要求された場合

    { kEventClassToolbar, kEventToolbarCreateItemWithIdentifier },
    // Toolbarに特定の新しいアイテムが作成された場合

    { kEventClassToolbar, kEventToolbarCreateItemFromDrag }
    // Toolbarにドラッグ＆ドロップで新しいアイテムが作成された場合
};


static OSStatus ToolbarDelegate( EventHandlerCallRef inCallRef,
                                 EventRef inEvent, void* inUserData )
{
    OSStatus            result = eventNotHandledErr;
    CFMutableArrayRef   array;
    CFStringRef         identifier;

    switch ( GetEventKind( inEvent ) )
    {
        case kEventToolbarGetDefaultIdentifiers: // ディフォルトアイテム

            GetEventParameter( inEvent, kEventParamMutableArray,
                               typeCFMutableArrayRef, NULL,
                               sizeof( CFMutableArrayRef ), NULL, &array );
             // Core FoundationのMutable Arrayを得てこれにアイテム情報を付加する

            GetToolbarDefaultItems( array ); // 自作ルーチン （後述参照）
            result = noErr;
            break;

        case kEventToolbarGetAllowedIdentifiers: //  構成変更用アイテム

            GetEventParameter( inEvent, kEventParamMutableArray,
                               typeCFMutableArrayRef, NULL,
```

```
                                    sizeof( CFMutableArrayRef ), NULL, &array );
    // Mutable Arrayを得てこれに構成変更用シートのアアイテム情報を付加する

    GetToolbarAllowedItems( array ); // 自作ルーチン
    result = noErr;
    break;

case kEventToolbarCreateItemWithIdentifier: // アイテムの新規作成
    {
        HIToolbarItemRef        item;
        CFTypeRef               data = NULL;

        GetEventParameter( inEvent,
                           kEventParamToolbarItemIdentifier, typeCFStringRef,
                           NULL, sizeof( CFStringRef ), NULL, &identifier );
                    // Toolbarアイテムの識別子を得る

        GetEventParameter( inEvent,
                           kEventParamToolbarItemConfigData, typeCFTypeRef,
                           NULL, sizeof( CFTypeRef ), NULL, &data );
                    // Toolbarアイテムの設定用データを得る

        item = CreateToolbarItemForIdentifier( identifier, data );
                    // 両者からアイテムを作る自作ルーチン（後述参照）

        if ( item )
        {
            SetEventParameter( inEvent, kEventParamToolbarItem,
                       typeHIToolbarItemRef,sizeof(HIToolbarItemRef ), &item );
                       // 作成されたToolbarアイテムの内容をシステムへ知らせる
            result = noErr;
        }
    }
    break;

case kEventToolbarCreateItemFromDrag: // アイテムのドラッグ＆ドロップ
    {
        HIToolbarItemRef        item;
        DragRef                 drag;
```

```
            GetEventParameter( inEvent, kEventParamDragRef, typeDragRef,
                                                NULL, sizeof( DragRef ), NULL, &drag );
                        // ドラッグ＆ドロップ・セッションのDragRefを得る

            item = CreateToolbarItemFromDrag( drag );
                        // DragRefからアイテムを作る自作ルーチン（後述参照）

            if ( item )
            {
                SetEventParameter( inEvent, kEventParamToolbarItem,
                        typeHIToolbarItemRef, sizeof( HIToolbarItemRef ), &item );
                         // 作成されたToolbarアイテムの内容をシステムへ知らせる
                result = noErr;
            }
        }
        break;
    }
    return result;
}
```

以下に示すように、Carbon EventとしてkEventToolbarGetDefaultIdentifiersが送られてきた時には、 パラメータとしてCore FoundationのMutable Arrayを受け取り、それにディフォルトのToolbarアイテムの情報（識別子など）を付加してやります。

```
static void GetToolbarDefaultItems( CFMutableArrayRef array )
{
   CFArrayAppendValue( array, CFSTR( "com.apple.carbontoolbar.anchored" ) );
   CFArrayAppendValue( array, kHIToolbarSeparatorIdentifier );
   CFArrayAppendValue( array, CFSTR( "com.apple.carbontoolbar.permanent" ) );
   CFArrayAppendValue( array, kHIToolbarFlexibleSpaceIdentifier );
   CFArrayAppendValue( array, CFSTR( "MyCustomIdentifier" ) );
   CFArrayAppendValue( array, CFSTR( "com.apple.carbontoolbar.trash" ) );
}
```

・Toolbarアイテムの作成

通常、Toolbarアイテムを作成する時にはユニークな文字列である識別子（Identifier）を指定する必要があります。

（a）システムで定義された識別子（Identifiers）

HIToolbar API には、システムで定義されてたスタンダードな識別子が用意されています。こうした識別子のアイテムは、特別な実装処理をしなくても利用する事ができます。自分で定義した識別子が、こうしたタンダード識別子とコンフリクトしないように注意してください。以下がスタンダード識別子の一覧です。

kHIToolbarSeparatorIdentifier
・アイテムのグループ分けをするための細い縦ライン

kHIToolbarSpaceIdentifier
・固定幅のスペース（無地の領域）

kHIToolbarFlexibleSpaceIdentifier
・可変幅のスペース（無地の領域）

kHIToolbarCustomizeIdentifier
・構成変更用のシートウィンドウをオープンするためのアイテム

kHIToolbarPrintItemIdentifier
・プリント用のコマンドIDをCarbon Eventとして送付する

kHIToolbarFontsItemIdentifier
・フォント選択用フローティングウィンドウを開く（Apple Type Servicesを参照）

（b）Toolbarアイテムのアトリビュート

Toolbarアイテムには、その挙動を決定するためのアトリビュートを設定することが可能です。以下がアイテムに付加できるアトリビュートの一覧です。より詳しい内容についてはHIToolbar.hを参照してください。

kHIToolbarItemNoAttributes
・アトリビュートなし

kHIToolbarItemAllowDuplicates
・このタイプのアイテムを2つ以上同時に表示可能

kHIToolbarItemCantBeRemoved
・ユーザはこのアイテムをToolbarの外へドラッグできない

kHIToolbarItemAnchoredLeft
・このアイテムはToolbarの左サイドに固定表示され移動できない

kHIToolbarItemIsSeparator
・このアイテムはアイテム間のセパレータとして利用される

kHIToolbarItemSendCmdToUserFocus
・コマンドはユーザフォーカスされたViewへ送られる（Mac OS X 10.3以降）

kHIToolbarItemLabelDisabled
・アイテムをマウスクリックしても何も起こらない（Mac OS X 10.4以降）

kHIToolbarItemDisabled
・アイテムを使用不可に設定する（Mac OS X 10.4以降）

kHIToolbarItemSelected
・アイテムを選択状態に設定する（Mac OS X 10.4以降）

（c）識別子（Identifiers）からアイテムを作成する

以下は、Carbon EventとしてkEventToolbarCreateItemWithIdentifierを受け取った時の処理ルーチンです。もし、受け取った識別子から利用対象アイテムでないと判断した時には、Carbon Event HandlerがeventNotHandledErrを返すようにします。

```
static HIToolbarItemRef CreateToolbarItemForIdentifier( CFStringRef identifier,
                                                        CFTypeRef configData )
{
    HIToolbarItemRef        item = NULL;

    if ( CFStringCompare( CFSTR( "com.apple.carbontoolbar.permanent" ),
                identifier, kCFCompareBackwards ) == kCFCompareEqualTo )
            // 文字列を比較して追加する対象の識別子かどうかを判断する
    {
        if ( HIToolbarItemCreate( identifier, kHIToolbarItemCantBeRemoved,
                        &item ) == noErr )
```

```
        // 削除可能のアトリビュートを付加してToolbarアイテムを作成する

    {
        IconRef       icon;
        MenuRef       menu;

        GetIconRef( kOnSystemDisk, kSystemIconsCreator, kFinderIcon,
                    &icon );
        // FinderアイコンのIconRefを得る
        HIToolbarItemSetLabel( item, CFSTR( "Can't Remove Me" ) );
        // アイテムにラベルを設定する
        HIToolbarItemSetIconRef( item, icon );
        // アイテムにアイコンを設定する
        HIToolbarItemSetCommandID( item, 'shrt' );
        // アイテムにコマンドID（'shrt' ）を設定する
        ReleaseIconRef( icon );
        // アイコンをリリースする
        menu = NewMenu( 0, "\p" );
        // タイトルなしのメニューを作成する（Menu Manager API）
        AppendMenuItemTextWithCFString( menu, CFSTR( "Item 1" ), 0, 0,NULL);
        AppendMenuItemTextWithCFString( menu, CFSTR( "Item 2" ), 0, 0,NULL);
        AppendMenuItemTextWithCFString( menu, CFSTR( "Item 3" ), 0, 0,NULL);
        // メニューに3つのメニュー項目を追加する
        HIToolbarItemSetMenu( item, menu );
        // メニューをアイテムに追加する
        ReleaseMenu( menu );
        // メニューをリリースする
    }
  }
  if ( CFStringCompare( CFSTR( "MyCustomIdentifier" ),  identifier,
                                kCFCompareBackwards ) == kCFCompareEqualTo )
        // 文字列を比較して追加する対象の識別子かどうかを判断する
  {
     item = CreateMyButtonToolbarItem( CFSTR( "MyCustomIdentifier" ));
        // カスタムToolbarアイテムを作成する自作ルーチン（後述参照）
  }
  return item;
}
```

（d）カスタムToolbarアイテムを作成する

カスタムToolbarアイテムを作成する場合には、そのアイテムをToolbarへ登録する前処理としてHIObjectのサブクラスとしてレジスト（登録）しておき、そのインスタンスを作成するためにCarbon Event Handlerによるコンストラクタを用意する必要があります。これは、HIViewモデルにおいてカスタムコントロールを作成する場合とまったく同じ処理となります。カスタムアイテムのレジストや作成について、より詳しい情報が得たい場合には、別ドキュメントの「HIView Programming Guide」を参照してください。

```
#define kMyButtonToolbarItemClassID CFSTR("com.moof.buttontoolbaritem")
                                                    //カスタムToolbarアイテムのクラス

const EventTypeSpec buttonEvents[] =     // コンストラクタ用イベントリスト
{
    { kEventClassHIObject, kEventHIObjectConstruct },  // インスタンス作成
    { kEventClassHIObject, kEventHIObjectInitialize },     // インスタンス初期化
    { kEventClassHIObject, kEventHIObjectDestruct },    // インスタンス削除
    { kEventClassToolbarItem, kEventToolbarItemCreateCustomView }
};                                                   // カスタムボタンのViewを作成

onst EventTypeSpec pushButtonEvents[] = // コントロール用イベントリスト
{
    { kEventClassControl, kEventControlGetSizeConstraints}
};                                                   // コントロールサイズ制限を設定

struct MyButtonToolbarItem                       // インスタンスデータ
{
    HIToolbarItemRef        toolbarItem;        // カスタムアイテム
};
typedef struct MyButtonToolbarItem MyButtonToolbarItem;

void RegisterMyButtonToolbarItemClass() // カスタムアイテムのレジスト
{
    static bool sRegistered;

    if ( !sRegistered ) // アプリケーション内で一度だけレジストとすればOK
    {
        HIObjectRegisterSubclass( kMyButtonToolbarItemClassID,
                        kHIToolbarItemClassID, 0, MyButtonToolbarItemHandler,
```

```
                    GetEventTypeCount( buttonEvents ), buttonEvents, 0, NULL );
        // カスタムアイテムをkHIToolbarItemClassIDのサブクラスとしてレジスト
        // Carbon Event HandlerルーチンはMyButtonToolbarItemHandler()

        sRegistered = true;
    }
}

static OSStatus MyButtonToolbarItemHandler( EventHandlerCallRef inCallRef,
                                            EventRef inEvent, void* inUserData )
{
    OSStatus                    result = eventNotHandledErr;
    MyButtonToolbarItem*    object = (MyButtonToolbarItem*)inUserData;

    switch ( GetEventClass( inEvent ) )
    {
        case kEventClassHIObject:  // コンストラクタ用のCarbon Event

            switch ( GetEventKind( inEvent ) ) // インスタンスデータの確保
            {
              case kEventHIObjectConstruct:
                  {
                    HIObjectRef                 toolbarItem;
                    MyButtonToolbarItem*    item;

                    GetEventParameter( inEvent,  kEventParamHIObjectInstance,
                                       typeHIObjectRef, NULL, sizeof( HIObjectRef ),
                                                              NULL, &toolbarItem );
                   // インスタンスを作成するToolbarアイテムを得る
                    result = ConstructMyButtonToolbarItem(toolbarItem, &item );
                    // インスタンスデータを確保する自作ルーチン（後述参照）

                    if ( result == noErr )
                        SetEventParameter( inEvent, kEventParamHIObjectInstance,
                                           typeVoidPtr, sizeof( void * ), &item );
                    // インスタンスデータをセットする。以降、Carbon Event Handler
                    //  によるメソッド実行時にインスタンスデータを参照できる。

                  }
```

```
            break;

        case kEventHIObjectInitialize:  // インスタンスデータの初期化

            if (CallNextEventHandler(inCallRef, inEvent) == noErr )
                {
                // superclassへ初期化の機会を与えるために実行する
                 HIToolbarItemSetLabel( object->toolbarItem,
                                                CFSTR( "Press Me" ) );
                // カスタムアイテム用のヘルプタグを設定

                 HIToolbarItemSetHelpText( object->toolbarItem,
                                            CFSTR("Moof!"), NULL );
                // カスタムアイテム用のラベル（テキスト）を設定

                 result = noErr;
                }
            break;

        case kEventHIObjectDestruct:  // インスタンスデータの削除

            free (object);  // インスタンスデータとして確保したメモリ領域を開放
            result = noErr;
            break;
    }
    break;

case kEventClassToolbarItem:  // Toolbarの処理に対応したメソッド

    switch ( GetEventKind( inEvent ) )
    {
      case kEventToolbarItemCreateCustomView:
                                    // コントロールのサイズ（矩形枠）制限を設定
            {
                EventTargetRef  myButtonEventTarget;
                HIViewRef       myButton;
                Rect            myButtonRect = {0,0,20,80}; //ボタンサイズ

                CreatePushButtonControl(NULL, &myButtonRect,
```

```
                                                    CFSTR("Push!"), &myButton);
                    // プッシュボタンコントロールの作成

                    SetEventParameter (inEvent, kEventParamControlRef,
                                       typeControlRef, sizeof(myButton), &myButton);
                    // プッシュボタンコントロールのHIViewRefをパラメータとして渡す

                    myButtonEventTarget = GetControlEventTarget(myButton);
                    // レジスト用のイベントターゲットを得る

                InstallEventHandler(myButtonEventTarget, MyButtonEventHandler,
                                    GetEventTypeCount(pushButtonEvents),
                                    pushButtonEvents, NULL, NULL);
                    // プッシュボタンコントロール用のCarbon Event Handlerを登録

                    result = noErr;
                }
                break;
            }
            break;
    }
    return result;
}
```

以下は、カスタムアイテムのインスタンスデータを確保するためのルーチンです。

```
static OSStatus ConstructMyButtonToolbarItem( HIToolbarItemRef inItem,
                                              MyButtonToolbarItem** outItem )
{
    MyButtonToolbarItem*        item;
    OSStatus                    err = noErr;

    item = (MyButtonToolbarItem*)malloc( sizeof( MyButtonToolbarItem ) );
    // インスタンスデータのメモリ領域を確保

    require_action( item != NULL, CantAllocItem, err = memFullErr );
    item->toolbarItem = inItem;
    // 構造体のメンバーに対象となるToolbarのHIToolbarItemRefを代入
```

```
    *outItem = item;  // 確保したインスタンスデータを返す

    CantAllocItem:
    return err;
}
```

以下は、カスタムToolbarアイテムとして用意したプッシュボタンにインストールしたCarbon Event Handlerです。イベント種類は、kEventControlGetSizeConstraintsのみに対応しています。

```
static OSStatus MyButtonEventHandler( EventHandlerCallRef inCallRef,
                        EventRef inEvent, void* inUserData )
{
    switch ( GetEventKind( inEvent ) )
    {
        case kEventControlGetSizeConstraints:  //コントロールサイズの制限
            {
                HISize minBounds = {80,20};  // 最小サイズ（矩形枠）
                HISize maxBounds = {80,20};  // 最大サイズ（矩形枠）

                SetEventParameter (inEvent, kEventParamMinimumSize,
                                    typeHISize, sizeof(HISize), &minBounds);
                                    // 最小サイズをパラメータで渡す

                SetEventParameter (inEvent, kEventParamMaximumSize,
                                    typeHISize, sizeof(HISize), &maxBounds);
                                    // 最大サイズをパラメータで渡す

                  return noErr;         // プッシュボタンのサイズ変更は不可である
              }
              break;

        default:
            break;
     }
    return eventNotHandledErr;
}
```

以下は、カスタムToolbarアイテム（プッシュボタン）を作成する自作ルーチンです。

```
HIToolbarItemRef CreateMyButtonToolbarItem( CFStringRef inIdentifier)
{
    OSStatus                err;
    EventRef                event;
    UInt32                  options = kHIToolbarItemAllowDuplicates;
    HIToolbarItemRef    result = NULL;

    RegisterMyButtonToolbarItemClass();  //   カスタムアイテムクラスのレジスト
    err = CreateEvent( NULL, kEventClassHIObject, kEventHIObjectInitialize,
                                                GetCurrentEventTime(), 0, &event );
                                // パラメータを渡すためのイベントを作成する

    require_noerr( err, CantCreateEvent );
    SetEventParameter( event, kEventParamToolbarItemIdentifier,
                                typeCFStringRef,sizeof( CFStringRef ), &inIdentifier );
                                // アイテムの識別子をパラメータをとして設定

    SetEventParameter( event, kEventParamAttributes, typeUInt32,
                                                        sizeof( UInt32 ), &options );
                                // アイテムのアトリビュートをパラメータをとして設定

    err = HIObjectCreate( kMyButtonToolbarItemClassID, event,
                                                            (HIObjectRef*)&result );
                                // カスタムアイテムを作成する
    check_noerr( err );
    ReleaseEvent( event );  // イベントをリリースする

CantCreateEvent:
    return result;
}
```

（e）ドラッグ＆ドロップよりカスタムToolbarアイテムを作成する

ユーザがToolbar上に何らかのオブジェクトをドラッグした場合、それが何であるかを調べ、表示可能であるアイテムであれば、Toolbarに追加することが可能です。以下は、テキストのドラッグ＆ドロップによりURLアイテムを作成するルーチンです。

```
static HIToolbarItemRef CreateToolbarItemFromDrag( DragRef drag )
```

```
{
    UInt16                  i, itemCount;
    HIToolbarItemRef    result = NULL;

    CountDragItems( drag, &itemCount );  // ドラッグアイテムの個数を得る
    for ( i = 1; i <= itemCount; i++ )          // 個数分だけ処理をループ
    {
        DragItemRef         itemRef;
        FlavorFlags         flags;

        GetDragItemReferenceNumber( drag, i, &itemRef );
        // 指定された番号のDragItemRefを得る

        if ( GetFlavorFlags( drag, itemRef, 'TEXT', &flags ) == noErr )
        // ドラッグアイテムにテキストデータが含まれているかどうかを調べる
        {
            Size            dataSize;
            char            string[256];
            CFURLRef    url;

            dataSize = sizeof( string );  //テキストデータのサイズを得る
            GetFlavorData( drag, itemRef, 'TEXT', &string, &dataSize, 0 );
            // もし存在していればドラッグアイテムからテキストデータを抽出する

            url = CFURLCreateWithBytes( NULL, string, dataSize,
                        kCFStringEncodingMacRoman, NULL );
            // テキストデータからCFURLRefを作成する（Core Foundation API）

            result = CreateMyURLToolbarItem( CFSTR( "MyURLIdentifier" ), url );
            // 自作のアイテム作成ルーチンへ渡す

            CFRelease( url );  // CFURLRefをリリースする
            break;
        }
    }
    return result;
}
```

上記ルーチンで実行されている自作のCreateMyURLToolbarItem()ルーチンについて

は、先んじて説明したCreateMyButtonToolbarItem()とほとんど同様ですので、そちらを参照してください。ここではHIObjectCreate()を用いて、カスタムアイテムを作成するのですが、このサブクラスのコンストラクタ（Carbon Event Handler）で確保しているインスタンスデータと、その初期化（kEventHIObjectInitializeでの処理）ルーチンは以下の様になります。このルーチンを実行する前に、必ずCallNextEventHandler()を実行するようにしてください。

```
struct CustomToolbarItem  // URL表示カスタムアイテム用のインスタンスデータ
{
    HIToolbarItemRef      toolbarItem;  // アイテムを配置するToolbar
    CFURLRef              url;          // アイテムに設定するURL
};
typedef struct CustomToolbarItem CustomToolbarItem;

static OSStatus InitializeCustomToolbarItem( CustomToolbarItem* inItem,
                                                        EventRef inEvent )
{
    CFTypeRef       data;
    IconRef         iconRef;

    if ( GetEventParameter( inEvent, kEventParamToolbarItemConfigData,
                                    typeCFTypeRef, NULL, sizeof( CFTypeRef ),
                                                NULL, &data ) == noErr )
    // パラメータからURLを含んだデータを読み込む
    {
        if ( CFGetTypeID( data ) == CFStringGetTypeID() )
            inItem->url = CFURLCreateWithString( NULL, (CFStringRef)data, NULL );
            // データがCFStringとして渡された場合にはCFURLRefへ変換して代入
        else
            inItem->url = (CFURLRef)CFRetain( data );
            // そうでない場合は直接インスタンスデータに代入
    }
    else
    {
        inItem->url = CFURLCreateWithString( NULL,
                                        CFSTR( "http://www.apple.com" ), NULL );
      // パラメータからURLを読み込めない場合はディフォルトを設定する
    }
    HIToolbarItemSetLabel( inItem->toolbarItem, CFSTR( "URL Item" ) );
```

```
    // Toolbarアイテムにラベル（テキスト）を設定する

    if ( GetIconRef( kOnSystemDisk, kSystemIconsCreator, kGenericURLIcon,
                                                        &iconRef ) == noErr )
    // システムディフォルトのURLアイコンを得る
    {
        HIToolbarItemSetIconRef( inItem->toolbarItem, iconRef );
        // Toolbarアイテムにアイコンを設定する

        ReleaseIconRef( iconRef );  // アイコンをリリースする
    }
    HIToolbarItemSetHelpText( inItem->toolbarItem,
                                            CFURLGetString( inItem->url ),  NULL );
    // Toolbarアイテムにヘルプタグを設定する（内容はURLのテキスト）

    return noErr;
}
```

本ドキュメントの履歴

オリジナル2005年8月11日　要約2005年10月15日 v1.00