

本ドキュメントは、アメリカ macromedia 社サイト

<http://www.macromedia.com/support/documentation/en/flash/>

の **Flash JavaScript Dictionary** にある fl_jsapi.zip (**fl_jsapi.pdf**) をヒム・カンパニーの永井勝則が独自に訳したものです。

fl_jsapi.pdf は、

Introduction

Top-level functions

Objects

C-Level Extensibility

の 4 部から成りますが、そのうちの **Intoroduction** 部を訳しています。

MX



macromedia
FLASH[™]MX
2004

Flash JavaScript Dictionary

CHAPTER 1

イントロダクション

本サイトのこのドキュメントでは、読者は JavaScript や ActionScript に習熟し、関数、パラメータ、データ型といったプログラミング・コンセプトに慣れ親しんでいることを前提としています。またオブジェクトとプロパティを使って作業するというコンセプトも理解している必要があります。JavaScript の参考には、Netscape JavaScript ドキュメントをご覧ください。

Netscape DevEdge Onlineには、JavaScript Developer Centralサイト (<http://developer.netscape.com/tech/javascript/index.html>) があり、JavaScriptを理解する上で役立つドキュメントや記事が掲載されています。最も有益なリソースはCore JavaScript Guideです。

Macromedia Flash JavaScript APIの概略

ActionScript言語では、Macromedia Flash Player環境（つまり、SWFファイルの再生中）においてアクションを実行させるスクリプトを書くことができます。Flash JavaScript API(JSAPI)では、Flashオーサリング環境（つまり、Flashプログラムが開かれている間）においていくつかのアクションを実行させるスクリプトを書くことができます。ツール・パネルにツールを追加するコマンドやスクリプトのように振舞うスクリプトを書くことができます。これらのスクリプトはオーサリング過程を自動化する支援として使用できます。

Flash JSAPIは、Macromedia DreamweaverやMacromedia Fireworks JavaScript API（これらはNetScape JavaScript APIに基づきデザインされています）を模してデザインされています。Flash JSAPIはDocument Object Model(DOM)に基づいています。これはJavaScriptオブジェクトを使ってFlashドキュメントにアクセスできるようにするものです。Flash JSAPIでは、Netscape JavaScript APIの全エレメントに加えFlash DOMも導入しています。これらの追加されたオブジェクトやそのメソッド、プロパティは本ドキュメントに記載されています。Flashスクリプト内でネイティブなJavaScript言語の要素をどれでも使用できますが、Flashドキュメント内で意味の通るエレメントのみ効果が出ます。

Macromedia Flash MX 2004 Professionalが好みのテキスト・エディタを使ってFlash JavaScript(JSFL)ファイルを書いたり編集したりできます。Flash Professionalをお使いの場合は、デフォルトで拡張子.jsflがファイルに付きます。コマンド・メニューにスクリプトを表示させるには、以下のフォルダにそのJSFLファイルを保存します。

・ Windows 2000 or Windows XP: C:\Documents and Settings\user\Local Settings\ Application Data\Macromedia\Flash MX2004\language\Configuration\Commands

・ Windows 98: C:\Windows\Application Data\Macromedia\Flash MX 2004\language\Configuration\Commands

・ Mac OS X: Hard Drive/Users/userName/Library/Application Support/Macromedia/Flash MX 2004/ language/Configuration/Commands

ツールを作成するJSFLファイルはToolsフォルダに保存する必要があります。それは以下の場所にあります。

_ Windows 2000 or Windows XP:

C:\Documents and Settings\user\Local Settings\ Application Data\Macromedia\Flash MX2004\language\Configuration\Tools

_ Windows 98:

C:\Windows\Application Data\Macromedia\Flash MX 2004\language\Configuration\Tools

_ Mac OS X:

Hard Drive/Users/userName/Library/Application Support/Macromedia/Flash MX 2004/ language/Configuration/Tools

JSFLファイルが、XMLファイルなどの他のファイルと一緒に動作する場合は、それらのファイルはJSFLファイルと同じディレクトリに保存します。

また履歴・パネル内の1つかそれ以上のコマンドを選択し、履歴・パネルのコマンドとして保存ボタンをクリックするか、オプションのポップアップ・メニューからコマンドとして保存を選択することで、JSFLファイルを作成することもできます。コマンド(JSFL)ファイルはCommandsフォルダに保存されます。保存後ファイルを開き、他のスクリプト・ファイルと同じように編集できます。

スクリプトを実行するには、以下のどちらかを実行します。

- ・ コマンド>コマンド名を選択します
- ・ コマンド>コマンドの実行を選択し、実行するスクリプトを選択します

Flashツール・パネルにJSFLファイルに提供されるツールを追加するには

- 1 . Toolsフォルダにツールとその他関係するファイルのJSFLファイルをコピーします。
- 2 . 編集>ツールパネルのカスタマイズを選択(Windows)、またはFlash>ツールパネルのカスタマイズを選択(Macintosh)します。
- 3 . 使用可能なツールのリストにツールを追加します。
- 4 . OKをクリックします。

ActionScriptファイルに独立したJSAPIコマンドを埋め込むこともできます。これは*Flash MX 2004 ActionScript Language Reference*に記載されているMMExecute()を使用します。しかし、MMExecute()コマンドは、コンポーネントのプロパティ・インスペクタやオーサリング環境でのSWFパネルといったカスタム・ユーザー・インターフェイスの状況で使用するときのみ有効になります。ActionScriptから呼ばれても、Flash Playerやオーサリング環境外では、JSAPIコマンドは無効になります。

JSAPIにはまたJavaScriptとカスタムCコードの組み合わせを使用することで機能を拡張できるメソッドが含まれています。さらに詳しい情報はChapter 4 “C-Level Extensibility”をご覧ください。

Flash JavaScriptオブジェクトにはプロパティとメソッドが含まれます。真偽値や整数、配列、浮動小数といった基本型や、色、オブジェクト、点、矩形ストリングなどのデータ参照型を定義するプロパティは、オブジェクトを記述するのに使用されます。メソッドはそのオブジェクトの関数を実行するのに使用されます。オブジェクトのプロパティやメソッドにアクセスするには、ドット表記を使用します。またほとんどのオブジェクトにはgetProperty()とsetProperty()メソッドが備わっていて、指定したプロパティの値を取得したり設定したりできます。ほとんどのメソッドは、そのメソッドの異なるオプションを特定するために使用されるパラメータをとります。

FlashにおけるJavaScriptインタープリタはMozilla SpiderMonkeyエンジン・バージョン1.5で、webサイト<http://www.mozilla.org/js/spidermonkey> で公開されています。SpiderMonkeyはMozilla.orgによって開発されたJavaScript言語の2つのリファレンス・インプリメンテーションの1つで、Mozillaブラウザに搭載されているのと同じエンジンです。

SpiderMonkeyはECMA-262仕様で定義されたコアJavaScript言語の全てを実行します。ECMA-262第4版に全面的に従ったものです。ECMA-262仕様の部分でないブラウザの特定のホスト・オブジェクトのみがサポートされません。

NetscapeのJavaScriptドキュメント・サイト (<http://devedge.netscape.com/central/javascript/>) の “Core JavaScript” セクションの全てはFlash JavaScriptインタープリタ適用されています。 ”Client-Side JavaScript”セクションの全ては適用されていません。それはそのセクションがブラウザ環境に適用されるのみであるからです。(意味不明)

SpiderMonkeyはまた、Fireworks MX 2004、Dreamweaver MX 2004、Director MX 2004、Flash Communication Server MXでも使用されています。

Flash Document Object Model

Flash JavaScript APIのDOMは、トップレベル関数（25Pの“Top-level functions”をご覧ください）とトップレベルflashオブジェクトから成ります。Flashオブジェクトは、Flashオーサリング環境が開かれているときは常に存在するので、スクリプトでいつでも利用できます。このオブジェクトを参照するときは、flashまたはflを使用します。例えば、開いているファイルを全て閉じるためには、以下のステートメントのどちらかを使います。

```
flash.closeAll();
```

```
fl.closeAll();
```

flashオブジェクトは以下のchildオブジェクトを含みます。

オブジェクト	アクセス方法
componentsPanelオブジェクト	componentsPanelオブジェクトにアクセスするには、fl.componentsPanelを使用します。このオブジェクトは、Flashオーサリング環境のComponentsパネルに相当します。
Documentオブジェクト	開いているドキュメントの配列を取得するにはfl.documentsを使用します。特定のドキュメントにアクセスするには、fl.documents[index]を使用します。現在のドキュメント（フォーカスの当たっているもの）にアクセスするには、fl.getDocumentDOM()を使用します。
drawingLayerオブジェクト	drawingLayerオブジェクトにアクセスするには、fl.drawingLayerを使用します。
Effectオブジェクト	Flashがスタートするとき登録される効果ディスクリプターに相当する効果の配列を得るには、fl.effectsを使用します。特定の効果にアクセスするにはfl.effectsを使用します。現在の効果が適用されている効果ディスクリプターにアクセスするには、fl.activeEffectを使用します。
Mathオブジェクト	Mathオブジェクトにアクセスするには、fl.Mathを使用します。
outputPanelオブジェクト	OutputPanelオブジェクトにアクセスするには、fl.outputPanelを使用します。このオブジェクトは、Flashオーサリング環境の出力パネルに相当します。
Toolsオブジェクト	fl.toolsはtoolObjsプロパティを持つオブジェクトです。toolObjsプロパティはtoolObjオブジェクトの配列です。各toolObjオブジェクトは、Flashツール・パネルのツールを表します。
ToolObjオブジェクト	全toolオブジェクトの配列を得るにはfl.tools.toolObjsを使用します。現在アクティブなtoolオブジェクトにアクセスするには、fl.tools.activeToolを使用します。

XMLUIオブジェクト	XML User Interface(XMLUI)オブジェクトにアクセスするには、 <code>fl.xmlui</code> を使用します。XMLUIオブジェクトは、XMLUIダイアログ・ボックスのプロパティを取得し設定する機能を提供します。
-------------	--

Document オブジェクト

トップ・レベルflashオブジェクトの重要なプロパティはdocumentsプロパティです。documentsプロパティは、オーサリング環境で現在開いているFLAファイルの1つを表すDocumentオブジェクトの配列を含みます。各Documentオブジェクトのプロパティは、FLAファイルが含むことのできるほとんどのエレメントを表します。よって、DOMの大部分はDocumentオブジェクトの子オブジェクトとプロパティから構成されています。

例えば、最初に開いたドキュメントを参照するには、ステートメント`flash.document[0]`か`fl.document[0]`を使用します。オーサリング環境で現在の作業をしている間、最初のドキュメントは、最初に開かれた、最初のFlashドキュメントです。最初に開かれたドキュメントが閉じられると、他の開かれているドキュメントのインデックスが1減ります。

あるドキュメントのインデックスを知るには、`fl.findDocumentIndex(nameOfDocument)`を使います。

現在フォーカスされているドキュメントにアクセスするには、ステートメント`flash.getDocumentDOM()`か`fl.getDocumentDOM()`を使用します。後者は本ドキュメントのほとんどの例で使用しているシンタックスです。

Documents配列であるドキュメントを見つけるには、配列を反復し、そのnameプロパティで各ドキュメントをテストします。

既出の表(P19のFlash Document Object Modelをご覧ください)にリストされていない、DOMの全オブジェクトは、Documentオブジェクトからアクセスできます。例えば、ドキュメントのライブラリにアクセスするには、ライブラリのオブジェクトを取得する、Documentオブジェクトのlibraryプロパティを使用します。

```
fl.getDocumentDOM().library
```

ライブラリのアイテムの配列にアクセスするには、libraryオブジェクトのitemsプロパティを使用します。配列の各エレメントはItemオブジェクトです。

```
fl.getDocumentDOM().library.items
```

ライブラリのあるアイテムにアクセスするには、items配列のメンバーを指定します。

```
fl.getDocumentDOM().library.items[0]
```

言い換えれば、LibraryオブジェクトはDocumentオブジェクトの子であり、ItemオブジェクトはLibraryオブジェクトの子です。

アクションのターゲットを指定する

指定しない限り、メソッドは現在のフォーカスか選択に影響を与えます。例えば、以下のスクリプトは、オブジェクトを指定していないので、現在の選択のサイズを2倍にします。

```
Fl.getDocumentDOM().scaleSelection(2,2)
```

いくつかのケースでは、Flashドキュメント内の特に現在選択されているアイテムをアクションのターゲットにしたいときもあります。そうするには、Document.selectionプロパティのもつ配列を使用します。配列の最初の要素は、現在選択されているアイテムを表します。以下に示すのはその例です。

```
Var accDescription = fl.getDocumentDOM().selection[0].description;
```

以下は、現在の選択でなく、ステージ上の最初の要素のサイズを倍にし、配列element内に保持するスクリプトです。

```
var element = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
if (element) {
    element.width = element.width*2;
    element.height = element.height*2;
}
```

またステージ上の全要素に対し繰り返しをさせて、指定した量だけ幅と高さを減らすこともできます。以下はその例です。

```
var elementArray = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements;
for (var i=0; i < elementArray.length; i++) {
    var offset = 10;
    elementArray[i].width -= offset;
    elementArray[i].height -= offset;
}
```

}

DOM構造のまとめ

以下のリストは、DOM構造のあらましです。各行の初めの数字は、オブジェクトのレベルを表しています。例えば、“03”で始まるオブジェクトは、次の上位“02”のオブジェクトの子で、“02”は次の上位“01”のオブジェクトの子です。

親オブジェクトのプロパティを指定することのできるオブジェクトもあります。例えば、`document.timelines` プロパティは、Timelineオブジェクトの配列を含みます。これらのプロパティは以下のあらましにも記されています。

オブジェクトには、他のオブジェクトの子ではなく、他のオブジェクトのサブクラスであるというオブジェクトもあります。別のオブジェクトのサブクラスであるオブジェクトは、他のオブジェクト（スーパークラス）のメソッドやプロパティに加え、それ自身のメソッドやプロパティを持ちます。サブクラスは、スーパークラスと同じ階層レベルを共有します。例えば、ItemはBitmapItemのスーパークラスです。その関係は、以下のあらましにも書かれています。

01 Top-level functions

01 flash object

02 componentsPanel object

02 Document object (fl.documents array)

03 Matrix object

03 Fill object

03 Stroke object

03 library object

04 Item object (library.items array)

04 BitmapItem object (subclass of Item object)

04 folderItem object (subclass of Item object)

04 fontItem object (subclass of Item object)

04 SoundItem object (subclass of Item object)

04 SymbolItem object (subclass of Item object)

04 VideoItem object (subclass of Item object)

03 Timeline object (document.timelines array)

04 Layer object (timeline.layers array)

05 Frame object (layer.frames array)

- 06 Element object (frame.elements array)
 - 07 Matrix object (Element.matrix)
- 06 Instance object (abstract class, subclass of Element object)
- 06 BitmapInstance object (subclass of Instance object)
- 06 CompiledClipInstance object (subclass of Instance object)
 - 07 Parameter object (ComponentInstance.parameters)
- 06 EmbeddedVideoInstance object (subclass of Instance object)
- 06 LinkedVideoInstance object (subclass of Instance object)
- 06 SymbolInstance object (subclass of Instance object)
- 06 Text object (subclass of Element object)
 - 07 TextRun object (text.textRuns array)
 - 08 TextAttrs object (textRun.textAttrs array)
- 06 Shape object (subclass of Element object)
 - 07 Contour object (shape.contours array)
 - 08 HalfEdge object
 - 09 Vertex object
 - 09 Edge object
 - 07 Edge object (shape.edges array)
 - 08 HalfEdge objectThe PolyStar example 23
 - 09 Vertex object
 - 09 Edge object
 - 07 Vertex object (shape.vertices array)
 - 08 HalfEdge object
 - 09 Vertex object
 - 09 Edge object
- 03 ScreenOutline object
 - 04 Screen object (screenOutline.screens array)
 - 05 Parameter object (screen.parameters array)
- 02 drawingLayer object
 - 03 Path object
 - 04 Contour object
- 02 Effect object (fl.effects array)
- 02 Math object
- 02 outputPanel object
- 02 Tools object (fl.tools array)
 - 03 ToolObj object (tools.toolObjs array)

多角形 サンプル

本ドキュメントではPolyStar.jsflという名前のFlash JSAPIスクリプトのサンプルをご紹介します（http://www.macromedia.com/go/jsapi_info_en/でファイルをダウンロードできます）。このスクリプトはFlashツールパネルに表示される多角形ツールを複製します。PolyStar.jsflファイルを読むと、JSAPIを使ってどのように多角形ツールをこしらえたか分かります。コード行が行っていることを説明する詳細なコメントが書かれているのです。このファイルを読んで、JSAPIの使用方法の理解に役立ててください。

Flash MX 2004には、PolyStar.jsflの初期バージョンが含まれていて、アップデートされたPolyStar.jsflファイルを使用するには、それを削除する必要があります。

Flash MX 2004によってインストールされたPolyStar.jsflの初期バージョンを削除するには

- 1 . 編集 > ツールパネルのカスタマイズ (Windows) か、Flash < ツールパネルのカスタマイズ (Macintosh) を選択します。
- 2 . ツールパネルのカスタマイズ・ダイアログボックスで、ダイアログボックスの左側にある矩形ツールをクリックします。
矩形ツールと多角形ツールがダイアログボックスの右側の現在の選択リストに表示されます。
- 3 . 現在の選択リストで多角形ツールを選択します。
- 4 . 削除をクリックします。
- 5 . OKをクリックします。
- 6 . Flashを終了させます。
- 7 . 1 7 Pの“ Macromedia Flash JavaScript APIの概略 ”にリストされている、適切なToolsフォルダからPolyStar.jsflファイルだけを削除します。PolyStar.xmlとPolyStar.pngファイルは、後でインストールする新しいPolyStar.jsflファイルが必要とします。Flashをリスタートさせると、多角形ツールはもうツールパネルのカスタマイズ・ダイアログボックスには表示されません。

アップデートされたPolyStarサンプル・ファイルをインストールするには

- 1 . 新しいPolyStar.jsflファイルをToolsフォルダにコピーします。このフォルダにあるPolyStar.xmlとPolyStar.pngファイルは新しいPolyStar.jsflファイルが必要とします。
- 2 . Flashをリスタートさせます。
- 3 . 編集 > ツールパネルのカスタマイズ (Windows) か、Flash > ツールパネルのカスタ

- マイズ(Macintosh)を選択します。使用可能なツールのリストにPolyStarがあるはずですが。
- 4 . ツールパネルのカスタマイズ・ダイアログボックスの左側にある矩形ツールをクリックします。矩形ツールが、ダイアログボックス右の現在の選択リストに現れるはずですが。
 - 5 . 使用可能なツールのリストで、PolyStarツールを選択します。
 - 6 . 追加をクリックします。
 - 7 . OKをクリックします。
- 多角形ツールが矩形ツール・ポップアップ・メニューに現れるようになります。

```
//PolyStar.jsfl
```

```
////////////////////////////////////
```

```
// define some variables available to the
```

```
// scope of this tool
```

```
////////////////////////////////////
```

```
var nSides = 5;
```

```
var pointParam = 0.5;
```

```
var style = "polygon";
```

```
var thePolygon = new Array;
```

```
var didDrag = false;
```

```
// the values for polygonStyle
```

```
var POLYGON = "polygon";
```

```
var STAR = "star";
```

```
////////////////////////////////////
```

```
// function configureTool
```

```
// Flash callback function.
```

```
// Each tool must have a configureTool function.
```

```
// This function is called by Flash when the
```

```
// application launches
```

```
////////////////////////////////////
```

```
function configureTool()
```

```

{
    // Set the standard tool information
    theTool = fl.tools.activeTool;
    theTool.setToolName("polystar");
    theTool.setIcon("PolyStar.png");
    theTool.setMenuString("PolyStar Tool");
    theTool.setToolTip("PolyStar Tool");
    theTool.setOptionsFile( "PolyStar.xml" );

    // This tool uses the Shape property inspector
    theTool.setPI( "shape" );
}

```

```

////////////////////////////////////
// function notifySettingsChanged
// Flash callback function
// Called when the tool options are
// changed by the user
////////////////////////////////////
function notifySettingsChanged()
{
    // the current values of the properties
    // for this tool are held by the activeTool object.
    theTool = fl.tools.activeTool;

    // update our local values of the properties
    // from the active tool object.
    // These properties are defined in the xml file for the tool.
    // Minimum and maximum values for these properties are set in the
    // xml file, so we accept the values without checking.
    nSides      = theTool.nSides;
    pointParam = theTool.pointParam;
    style       = theTool.style;
}

```



```

// Flash callback function
// Called when the user presses the mouse button
// in the workspace when this tool is active
////////////////////////////////////
function mouseDown()
{
    // start drawing of object
    fl.drawingLayer.beginDraw();

    // set the flag if the cursor moves "enough"
    didDrag = false;
}

////////////////////////////////////
// function mouseMove
// Flash callback function
// Called when the user moves the mouse
////////////////////////////////////
function mouseMove(mouseLoc)
{
    // only handle the callback if the mouse button is down
    if (fl.tools.mouseIsDown)
    {
        // check how much the mouse has moved since the pen went down
        var pt1 = fl.tools.penDownLoc;
        var pt2 = fl.tools.snapPoint( mouseLoc );
        var dx = pt1.x - pt2.x;
        var dy = pt1.y - pt2.y;
        if (dx < 0)  dx = -dx;
        if (dy < 0)  dy = -dy;

        // constrain with the shift key
        if (fl.tools.shiftIsDown)
        {
            // The following code will put a corner of the polygon/star

```

```

// pointing directly up/down/left/right depending on the
// position of the mouse relative to the center.

var radSq = dx*dx + dy*dy;
var rad = radSq > 0.01 ? Math.sqrt( radSq ) : 0.0;

var dTheta = Math.PI/nSides;
var angle = Math.PI/2.0;

// put a point near the cursor
if (Math.abs(dx) > Math.abs(dy))
{
    if (pt2.x < pt1.x)
        angle += Math.PI/2.0;
    else
        angle -= Math.PI/2.0;
}
else
{
    if (pt2.y < pt1.y)
        angle = -angle;
}

pt2.x = pt1.x + rad*Math.cos( angle );
pt2.y = pt1.y + rad*Math.sin( angle );
}

if ((dx > 2) || (dy > 2))
{
    // if the user does not move the mouse enough
    // no shape is created when the button is released.
    // use the following flag to indicate that the shape should be created.
    didDrag = true;

    // build the array of points defining the new polygon
    // based on the current values of the points.

```

```

        buildPolygonObj(pt1, pt2, thePolygon);

        // draw the current polygon
        // The drawing is surrounded by calls to
        // beginFrame and endFrame. Doing this causes Flash
        // to erase the previous polygon and draw the new one.
        fl.drawingLayer.beginFrame();
        drawPolygonObj( thePolygon );
        fl.drawingLayer.endFrame();
    }
}

////////////////////////////////////
// function mouseUp
// Flash callback function
// Called when the user releases the mouse button
////////////////////////////////////
function mouseUp()
{
    // notify Flash that we are finished drawing
    fl.drawingLayer.endDraw();

    // if the mouse moved enough create the shape on the stage
    if (didDrag)
    {
        // create a path object from the array of points
        var path = polygonToPath( thePolygon );

        // create the shape on the stage
        path.makeShape();
    }
}

```

```

////////////////////////////////////
// function buildPolygonObj
// Function to array with points that define the
// polygon based on the values of the arguments
////////////////////////////////////
function buildPolygonObj( pt1, pt2, thePolygon )
{
    // the point given by the first argument is used as the center point
    var ctr = new Object;
    ctr.x = pt1.x;
    ctr.y = pt1.y;

    // calculate the radius
    var rad = fl.Math.pointDistance(pt1, pt2);

    // find the angle between points
    var doStar = (style == STAR);
    var dTheta = 2.0*Math.PI/nSides;

    var param = pointParam;
    if (param < 0.1) param = 0.1;
    if (param > 0.9) param = 0.9;

    // The locations of the points of the polygon
    // are calculated by rotating points about the origin.
    var x = pt2.x - pt1.x;
    var y = pt2.y - pt1.y;
    var cs = Math.cos(dTheta);
    var sn = Math.sin(dTheta);
    thePolygon[0] = x;
    thePolygon[1] = y;
    var index = 2;
    for (var i=0; i<nSides; i++)
    {
        // apply a rotation to the point
        // The last point is taken from the first point.

```

```

var xtmp, ytmp;
if (i == (nSides-1))
{
    xtmp = thePolygon[0];
    ytmp = thePolygon[1];
}
else
{
    xtmp = x*cs - y*sn;
    ytmp = x*sn + y*cs;
}

// if we are building a star calculate an additional
// point that lies on the line joining the origin
// with the midpoint of the line segment between
// the current and previous polygon points.
if (doStar)
{
    thePolygon[index] = param*0.5*(xtmp + x)
    thePolygon[index+1] = param*0.5*(ytmp + y)
    index += 2;
}

// update x and y to the new values
x = xtmp;
y = ytmp;

// offset to the center
thePolygon[index] = xtmp;
thePolygon[index+1] = ytmp;
index += 2;
}

// adjust the length of the array
thePolygon.length = index;

```

```

// offset to the center
for (var i=0; i<thePolygon.length; i += 2)
{
    thePolygon[i ] += ctr.x;
    thePolygon[i+1] += ctr.y;
}

return;
}

```

```

////////////////////////////////////
// function transformPoint
// Apply a matrix transformation on the given point
////////////////////////////////////
function transformPoint( pt, mat )
{
    var x = pt.x*mat.a + pt.y*mat.c + mat.tx;
    var y = pt.x*mat.b + pt.y*mat.d + mat.ty;

    pt.x = x;
    pt.y = y;

    return;
}

```

```

////////////////////////////////////
// function drawPolygonObj
// draw a polygon defined by the points in
// the input array using the drawingLayer
////////////////////////////////////
function drawPolygonObj( thePolygon )
{
    if (thePolygon.length != 0)
    {

```

```

// define 2 points to hold the transformred values
var tmpPt  = new Object;
var tmpPt2 = new Object;
tmpPt.x = thePolygon[0];
tmpPt.y = thePolygon[1];

// if the polygon is being drawn inside a symbol currently
// being edited in Flash, we must transform to document
// space using the view matrix
var viewMat = fl.getDocumentDOM().viewMatrix;
transformPoint(tmpPt,  viewMat);

// set the pen position to the location of the first point
fl.drawingLayer.moveTo( tmpPt.x,  tmpPt.y );

// draw the segments of the polygon
var index = 3;
while (index < thePolygon.length)
{
    // transform to document space
    tmpPt.x  = thePolygon[index-1];
    tmpPt.y  = thePolygon[index];
    transformPoint(tmpPt,  viewMat);

    // draw the next line
    fl.drawingLayer.lineTo(tmpPt.x,  tmpPt.y,  tmpPt2.x,  tmpPt2.y);
    index += 2;
}
}
}

////////////////////////////////////
// function polygonToPath
// Convert the array of polygon points to a path object
////////////////////////////////////

```

```
function polygonToPath( thePolygon )
{
    // allocate a path
    var path = fl.drawingLayer.newPath();

    // add the segments
    path.addPoint(thePolygon[0], thePolygon[1]);
    var index = 3;
    while (index < thePolygon.length)
    {
        path.addPoint( thePolygon[index-1], thePolygon[index] );
        index += 2;
    }

    return path;
}
```