Custom Transforms

Introduction

Mako `ITransforms` provide a method of applying common operations on DOM objects such as brushes, colors, colorspaces, glyphs or paths. They can operate on individual nodes or entire trees.

For example, there are many built-in transforms in Mako for:

- Image downsampling (`IImageDownsamplerTransform`)
- Color conversion (`IColorConverterTransform`)
- Color simplification (`IComplexColorSimplifierTransform`)
- Image merging (`IImageMergerTransform`)
- Remove non-visible Optional Content (`IOptionalContentFixerTransformPtr`)

Thus, transforms simplify complex operations to be applied to a number of DOM objects in one go.

A custom transform enables a Mako SDK developer to take advantage of the `ITransforms` framework for their own purposes.

Custom Transform header

A framework to enable development of custom transforms is provided as a header, `customtransform.h`.

It's now part of the standard Mako 5.x distribution, but this was not the case for Mako 4.x. The headers for those versions can be downloaded from the link shown on the right of this page. Note that there are two versions, one for pre-Mako 4.8 and another for Mako 4.8.

A custom transform implements call backs for the DOM objects listed below. All you need do is override the cases you are interested in, providing one or more methods for actually doing the work

For example, let's say you wanted to find out information for every image in a document. You could write a custom transform to override `transformImageBrush()`. Your code will be called every time an imagebrush (`IDOMImageBush`) is encountered, and from there get to the image (`IDOMImage`) and its frame (`IDOMFrame`) to obtain all the information you need - size, resolution, colorspace etc. You would run the transform on a page and repeat for all pages in a document.

A more likely scenario is one where you want to change the object in some way. Your implementation can do this, returning the updated object or a NULL if you want it removed from the DOM tree.

IDOM object	Description	IDOM object	Description
IAnnotationAppearance	Annotation appearance	**IDOMForm**	The children of this node type comprise the contents of a PDF/PS style form. (XForm)
IDOMColor	Color	**IDOMFormInstance**	A node whose content must be an IDOMForm
IDOMColorSpace	Color space	**IDOMBrush**	The parent class for the many brush types. Brushes are used to fill paths, ie a defined geometric region on the page
IDOMImage	Base class describing an image	**IDOMSolidColorBrush**	A solid color brush is used to fill a path with a solid color

Custom Transform headers

Mako 4.7 or earlier	Ma
customtra...ko4.7.zip	

IDOMNode	Base class for the many DOM node types	**IDOMGradientBrush**	The parent class for linear and radial gradient brushes
IDOMFixedPage	The contents of a page	**IDOMLinearGradientBrush**	A linear gradient brush is used to specify a gradient along a vector
IDOMGroup	A DOM node representing Group Elements. A group of IDOMNodes that share common properties such as a clipping path or render transform	**IDOMRadialGradientBrush**	A radial gradient brush defines an ellipse to be filled with a gradient
IDOMCharPathGroup	A DOM node representing Group Elements that consist of stroked text	**IDOMVisualBrush**	A visual brush is used to fill a path with a vector drawing
IDOMTransparencyGroup	A DOM node representing Group Elements that share common transparency settings. Analogous to PDF Transparency Groups.	**IDOMImageBrush**	An image brush is used to fill a path with an image
IDOMCanvas	A canvas is a special form of an isolated, non-knockout, normal blended transparency group. It groups elements of a page together.	**IDOMTilingPatternBrush**	A tiling pattern brush is used to fill a path with a PS-style tiling pattern.
IDOMGlyphs	Glyphs nodes are used to represent a run of uniformly formatted text from a single font	**IDOMShadingPatternBrush**	A shading pattern brush is used to fill a defined geometric region with a PS-style shading pattern.
IDOMFont	Font base class	**IDOMSoftMaskBrush**	A soft mask brush provides a way of representing a PDF-style soft mask in its entirety. It will contain a suitable `IDOMTransparency` group as well as the necessary soft mask details
IDOMPathNode	A path node specifies geometry that can be filled with a brush	**IDOMMaskedBrush**	A masked brush describes a generalization of a masked image
IDOMVisualRoot	Special node type used by XPS for the root node inside a visual brush (normally tiling patterns)	**IDOMNullBrush**	A null brush

CTransformState

Class for tracking the graphics state leading to the point where a transform is applied.

Consider for example the IImageDownsamplerTransform described elsewhere. In order to determine how to downsample an image, the transform needs to know how large the image will eventually be. The CTransformState provides this information by providing the combined transform that applies to the image based on the RenderTransforms of all the nodes entered leading to the point where the image is actually encountered.

Other transforms need access to other information, such as the approximate clip area, the current group color space, the renderingIntent (if present), the current antialiasing mode (edge mode) and/or how a brush is used.

CTransformState::transformPriv

New to Mako 4.8.0, this member can be used to track extra information needed by the transform process. Use it as you wish.

Implementing a custom transform

This **example** of a custom transform implementation converts glyphs to paths.

Note that:

- You need only to implement transforms for those objects you are interested in examining or changing
- Returning the default `genericImplementation` allows the transform to continue, possibly recursing further into the object
- The object you return will replace the object being transformed (and therefore if it's the same object, will have no effect)
- Returning a null or empty object will result in that object being removed from the tree

In the example, a path is returned which replaces the glyph object in the tree, ie the glyph object is removed and the path added.

Creating and calling the transform

It is instantiated by the calling program:

Creating the transform instance

```
                // Create a transform to convert glyphs to paths
                GlyphTransformImplementation implementation(jawsMako,
strokeWidth, solidBrush);
                ITransformPtr outliner = ICustomTransform::create
(jawsMako, &implementation, abort, true, true, true, true, true, true);
```

Once the instance exists, there is no need to create it again. Just call it as required. In the example it is called for each page:

Calling the transform

```
        // Apply the transform to every page
        for (uint32 pageNum = 0; pageNum < pageCount; pageNum++)
        {
                        IPagePtr page = document->getPage(pageNum);
                IDOMFixedPagePtr fixedPage = page->edit();

                        // Convert any glyphs
                        bool result = false;
                        outliner->transform(fixedPage, result);
        }
```

One advantage of custom transforms is that they can hide away complexity. Remember that you can implement call backs for as many objects types as you wish, in the one transform.

Object sharing

In general, if the custom transform callback has a “changed” parameter, then the object being passed in is not shared and can be edited at will, providing you set the “changed” result appropriately.

If the custom transform callback does not, then the object is potentially shared and you must make a clone before editing it. The internal transformation machinery detects changes if a different object is returned.

What are all those Booleans in the call to the transform?

They control cacheing behavior. Depending on the nature of the transform, it may not be necessary to process the same transform repeatedly. Setting one or more of these values to `false` may speed up a custom transform by allowing Mako to return a reference to a cached copy. Each relates to the properties of various object types, seen here in this excerpt from `customtransform.h`:

Custom transform call parameters

```
static JAWSMAKO_API ICustomTransformPtr create(const IJawsMakoPtr
&jawsMako,
                                               IImplementation
*implementation,
                                               const IAbortPtr &abort =
IAbortPtr (),
                                               bool dependsOnClipBounds =
true,
                                               bool dependsOnGroupSpace =
true,
                                               bool
dependsOnRenderingIntent = true,
                                               bool dependsOnTransform =
true,
                                               bool dependsOnBrushUsage =
true,
                                               bool dependsOnEdgeMode =
true,
                                               bool
dependsOnUncoloredTilingBrush = true);
```

We recommend you experiment with these to determine their applicability to your implementation.