T3D Awesomium Integration Pack

Welcome to the Torque 3D Awesomium Integration Pack. In this document we will go trough the steps of merging the pack with T3D. There's also a tutorial which will teach you the steps nessecary to create a shape from the editor.

Always make backups of your source files before attempting any merges.

www.stefanlundmark.com

https://github.com/StefanLundmark/Torque3D_Awesomium

Table of contents

1 GETTING STARTED 1
1.1 DOWNLOAD AWESOMIUM 1
1.2 PREPARING YOUR T3D DISTRIBUTION 1
2 CHANGES TO T3D 2
2.2 FIXES 2
2.2.1 Gui/Core/GuiCanvas.cpp 2
2.2.2 PlatformWin32/WinProcessControl.cpp 3
2.3 HOOKING UP THE PLAYER CLASS WITH THE PACK 4
2.3.1 T3D/Player.h 4
2.3.2 T3D/Player.cpp 4
2.4 HOOKING UP GUICROSSHAIRHUD WITH THE PACK 7
2.4.1 T3D/Fps/GuiCrossHairHud.cpp 7
2.5 ADDING OUR NEW OBJECT TYPE 8
2.5.1 T3D/ObjectTypes.h 8
2.6 COPYING THE CHANGED EDITOR SCRIPTS 8
2.7 NOTE ON GUIPROFILES 8
3 WRAPPING UP 9
4 AWSHAPE TUTORIAL 10
4.1 CREATING THE SHAPE 10
4.2 CREATING THE MATERIAL IN T3D 10
4.3 CREATING THE TEXTURE TARGET 11
4.4 EXECUTE THE SCRIPTS 12
4.5 PLACE THE SHAPE IN THE GAME 12

www.stefanlundmark.com

1 Getting started

1.1 Download Awesomium

Head over to www.awesomium.org and download their Release Candidate. This version of the pack was compiled against Awesomium 1.7.

1.2 Preparing your T3D distribution

Copy Engine/Source/Awesomium from the GitHub archive to your Engine/Source in your T3D directory.

Copy the following files to your T3D game directory where your binary
(Usually FPS Example.exe) is located:

- awesomium.dll
- awesomium_process.exe
- icudt.dll

Then you can open up your T3D project solution. Bring up the Properties for *FPS Example DLL* and go to Linker – Input. You'll need to link with *Awesomium.lib* here.

You'll also have to make sure the include files in Awesomium are used, so head over to C++ - General in the same dialog, and make sure one entry is pointing to Awesomium/Include in the Additional Include Directories input box.

Now we have to make some *simple* changes to T3D MIT before we can use the pack.
Don't forget to make backups of your source files if you haven't already!

2 Changes to T3D

2.2 Fixes

Depending on the version of T3D you're using, there are some fixes that are required for the pack to be stable.

2.2.1 Gui/Core/GuiCanvas.cpp

Find *GuiCanvas::getCursorPos()*. It should look like this if it's unpatched:

```
Point2I GuiCanvas::getCursorPos()
{
   Point2I p( 0, 0 );

   if( mPlatformWindow )
      mPlatformWindow->getCursorPosition( p );

   return p;
}
```

Replace it with this:

```
Point2I GuiCanvas::getCursorPos()
{
   Point2I p( 0, 0 );

   if( mPlatformWindow )
   {
      mPlatformWindow->getCursorPosition( p );
      // T3D Awesomium Integration Kit
      p = mPlatformWindow->screenToClient( p );
   }

   return p;
}
```

2.2.2 PlatformWin32/WinProcessControl.cpp

There is a bug in Awesomium which forces us to use a different method to force T3D to shutdown. Find *Platform::forceShutdown()*. It should look like this:

```
void Platform::forceShutdown(S32 returnValue)
{
   // Don't do an ExitProcess here or you'll wreak havoc in a multithreaded
   // environment.

   exit( returnValue );
}
```

Replace it with this:

```
void Platform::forceShutdown(S32 returnValue)
{
   // Don't do an ExitProcess here or you'll wreak havoc in a multithreaded
   // environment.

   // T3D Awesomium Integration Kit
   _exit( returnValue );
}
```

2.3 Hooking up the Player class with the pack

2.3.1 T3D/Player.h

Find *S32 mPredictionCount;* and put the following **below** it:

```
// T3D Awesomium Integration
F32 mAwesomiumZOffset;
```

Find *processTick()* and insert the following **above** it:

```
// T3D Awesomium Integration
virtual void processAwesomiumTick (const Move *move);
void advanceAwesomiumTime (F32 dt);
```

2.3.2 T3D/Player.cpp

Insert the following after the rest of the include statements at the top of the file:

```
// T3D Awesomium Integration
#include "Awesomium/AwShape.h"
#include "Awesomium/AwTextureTarget.h"
#include "Awesomium/AwManager.h"
```

Find *Player::Player()* and insert the following at the end of the method:

```
// T3D Awesomium Integration
mAwesomiumZOffset = 0.0f;
```

Find *Player::processTick()* and insert the following **above** it:

```
// T3D Awesomium Integration
void Player::processAwesomiumTick (const Move *move)
{
    if (isServerObject ())
        return;

    MatrixF mat;
    Point3F start, dir;
    getEyeBaseTransform (&mat, false);
    start = mat.getPosition ();
    mat.getColumn (1, &dir);
    Point3F end = start + (dir * AwManager::getRayLengthScale ());

    // Push back the start point a little bit incase we're
    // getting a little bit too close..
    start -= dir;

    RayInfo info;
    AwTextureTarget *target = NULL;
    AwShape *shape = NULL;
    if (getContainer ()->castRay (start, end, AwShapeObjectType, &info))
    {
        shape = (AwShape *)info.object;
        target = shape->processAwesomiumHit (start, end);
        if (target)
        {
            // Eat the mouse trigger so we can't fire, and make sure
            // Awesomium registers the event.
            shape->setIsMouseDown (move->trigger [0]);
            ((Move *)move)->trigger [0] = false;
        }
        else
            shape = NULL;
    }

    AwShape::setMouseInputShape (shape);
    AwTextureTarget::setMouseInputTarget (target);
}
```

Further down in *processTick()* find the following:

```
   // If we're not being controlled by a client, let the
   // AI sub-module get a chance at producing a move.
   Move aiMove;
   if (!move && isServerObject() && getAIMove(&aiMove))
      move = &aiMove;
```

Insert the following **below** the above mentioned snippit:

```
   // T3D Awesomium
   if (move)
      processAwesomiumTick (move);
```

We're almost done! Find *Player::advanceTime()* and insert the following **above** it:

```
// T3D Awesomium Integration
void Player::advanceAwesomiumTime (F32 dt)
{
   F32 dropAmount = 0.5f;
   F32 dropSpeed = AwManager::getImageDropSpeed ();

   // If we've got focus on a shape, we will keep process hits on
   // it to get fluid mouse movement.
   if (AwShape::getMouseInputShape () && dropSpeed > 0.0f)
   {
      mAwesomiumZOffset -= dropSpeed * dt;
      if (mAwesomiumZOffset < -dropAmount)
         mAwesomiumZOffset = -dropAmount;
   }
   else
   {
      mAwesomiumZOffset += dropSpeed * dt;
      if (mAwesomiumZOffset > 0.0f)
         mAwesomiumZOffset = 0.0f;
   }
}
```

Further down in *advanceTime()* add the following at the end of the method:

```
   // T3D Awesomium Integration
   advanceAwesomiumTime (dt);
```

Finally, find *Player::renderMountedImage()* and the following code:

```
         // See if we are pushed into a wall...
         Point3F start, end;
         smat.getColumn(3, &start);
         nmat.getColumn(3, &end);

         Point3F displace = (start - end) * mWeaponBackFraction;
```

Below it, add the following:

```
// T3D Awesomium Integration
displace.z += mAwesomiumZOffset;
```

2.4 Hooking up GuiCrossHairHud with the pack

2.4.1 T3D/Fps/GuiCrossHairHud.cpp

Insert the following after the rest of the include statements at the top of the file:

```
// T3D Awesomium Integration
#include "Awesomium/AwTextureTarget.h"
```

Find *GuiCrossHairHud::onRender()* and the following code:

```
ShapeBase* control = dynamic_cast<ShapeBase*>(conn->getControlObject());
if (!control || !(control->getTypeMask() & ObjectMask) || !conn>isFirstPerson())
   return;
```

Below the above, insert the following:

```
   // T3D Awesomium Integration
   // If we've got focus on a target, there's no reason to show the crosshair.
   if (AwTextureTarget::getMouseInputTarget ())
               return;
```

2.5 Adding our new object type

2.5.1 T3D/ObjectTypes.h

Find the enum called *SceneObjectTypes* and scroll down until the end of it. Insert the following as the last type:

```
// T3D Awesomium Integration
AwShapeObjectType = BIT( 23 ),
```

2.6 Copying the changed editor scripts

This assumes that you haven't already made any modifications to any of the editor scripts. If you have, you'll have to merge the modifications manually. They're very simple, so it should be trivial.

Copy the "tools" directory into your game directory. This will let you add AwShapes from inside the editor.

2.7 Note on GuiProfiles

Like with all GUI's in T3D; if you want input from the keyboard as text you'll have to create the proper GuiProfile for this and link it to the GuiControl. This is only needed for AwGuis.

```
if( !isObject( GuiAwesomiumProfile ) )
new GuiControlProfile( GuiAwesomiumProfile )
{
   textOffset = "4 2";
   autoSizeWidth = false;
   autoSizeHeight = true;
   justify = "left";
   tab = true;
   canKeyFocus = true;
   category = "Core";
};
```

The profile can be added to any .cs file that is executed by the engine, for example: core/art/gui/profiles.cs.

The profile itself must have canKeyFocus set to true or it won't accept keyboard input as text.

3 Wrapping up

Congratulations, you're now ready to use HTML5/Flash content in your game!

There is a barebones demonstration build included in Demo.rar. To launch, run *FPS Example.exe* from the demo. The HTML/JavaScript sources can be found in the Awesomium subdirectory of the demo.

There is JavaScript integration code in each .html file in the distribution. Some of these are responsible for making calls to TorqueScript. The other part of the integration is in **Functions.cs** where you can find TorqueScript code responsible for calling back to JavaScript.

You may also have to tweak the `$pref::Awesomium::RayLengthScale` property in your client's prefs.cs file if you want your shapes to be registered for input at long ranges.

Have fun!

4 AwShape Tutorial

This tutorial will go trough the process of creating a new AwShape pointing to a website of your choice. The demo is a good starting point, but this tutorial will assume the demo hasn't been used.

4.1 Creating the shape

You need to create a shape in whatever modelling program you're using. The material which you want to show the website on needs to be connected to T3D. We will do this in the next step, but it's a good idea to keep track of all the material names in the shape.

4.2 Creating the material in T3D

When you've got the shape ready, create a subdirectory in your game directory called Awesomium. All naming and locations are optional, obviously - just make sure you're consistant.

Inside the directory, create a new .cs file named Materials.cs. Open it up, and add the following to it:

```
singleton Material(MyMaterial)
{
            mapTo = "FrontMaterial";
            diffuseMap[0] = "#FrontMaterialTargetName";
            specular[0] = "0 0 0 1";
            specularPower[0] = "2";
            emissive [0] = true;
            translucentBlendOp = "None";
};
```

This is what it does:

- It creates a material named **MyMaterial**. The naming isn't important in this case, but it needs to be unique.
- **mapTo** maps the material to a material in your shape with the name **FrontMaterial**. This is important and needs to match whatever name you assigned your material in the modelling program.
- **diffuseMap[0]** is the diffuse color of the material. This is usually a texture of some sort, but in our case we want it to be linked to a dynamic texture which we update ourselves from Awesomium. This is why there's a # infront of the name. You can name this whatever you want, but it needs to be unique. In a later step you'll learn how to use it.
- The rest of the entries are optional but makes for a good material. Emissive in particar is good because you don't want shadows to darken a TV screen, for example. But if you want shadows, just remove it.

4.3 Creating the texture target

Now we need to create a texture target which will feed the material with its dynamic texture from Awesomium.

Inside the directory where you created Materials.cs, create another file called TextureTargets.cs. Open it up, and add the following to it:

```
singleton AwTextureTarget(MyTextureTarget)
{
    TextureTargetName = "FrontMaterialTargetName";
    OnGainMouseInputSound = "WebShapeOnGainMouseInputSound";
    OnLoseMouseInputSound = "WebShapeOnLoseMouseInputSound";
    StartURL = "http://www.mysite.com";
    Resolution = "1024 768";
    UseBitmapCache = true;
};
```

This is what it does:

- It creates a AwTextureTarget called **MyTextureTarget**. This is just like the material, not very important but needs to be unique.
- **TextureTargetName** is important and needs to match the diffuse map entry from your material. Note however that you do not use a # in this case as it's only used to tell the engine that the texture is from a target in the material definition.
- The two input sounds are optional. You can point these at any sound profile of your choice. These are all over the T3D examples with a lot of examples.
- **StartURL** specifies which URL the target should point to. This is where you specify your website, or a file. For files, take a look at the demo targets.
- **Resolution** specifies at what resolution the webpage should render. A lower resolution is faster and consumes less memory, but doesn't look as good and you might get issues with some sites that assume you're using a minimum resolution.
- **UseBitmapCache** specifies that the AwTextureTarget should save a cache on disk when it has rendered the first frame of the fully loaded website. The next time you load the site, it will be shown instantly (the cache will be used) avoiding any blinking that might occur when the player enters the level and the site hasn't been fully loaded yet. It's an optional feature.

4.4 Execute the scripts

The scripts are finished and ready to be used, but they need to be executed as well or T3D has no idea of their existence. To do this, open up **main.cs** in your game root and add the following at the end of the file:

```
exec ("Awesomium/Materials.cs");
exec ("Awesomium/TextureTargets.cs");
```

Close all files we've used and make sure they're saved.

4.5 Place the shape in the game

Start your game and head into the editor by pressing F11. Look at the topright corner and you'll see the **Scene Tree**. Press the Library button below, then Level, and Level once again. Double click the **Awesomium Shape** item, pick the shapefile you created in the first step and place it where you want it.

If everything went well, your website should now render on the material!