Creating New Post FX

In order to create a new post process effect, one must first create a class with the base *PostFX*
Please, see the following child of *PostFX*

```
class PFX_Copy : public PostFX
{
public:
        struct Settings
        {
                Settings();

                float TexScaleX ;
                float TexScaleY ;

                bool SkyOnly ;
        };

        typedef r3dTL::TArray< Settings > SettingsArr ;

        // construction/ destruction
public:
        explicit PFX_Copy( int BlendMode, int ColorWriteMask );
        ~PFX_Copy();

        // manipulation /access
public:
        void PushSettings( const Settings& settings ) ;

        // polymorphism
private:
        virtual void InitImpl();
        virtual void CloseImpl();
        virtual void PrepareImpl(     r3dScreenBuffer* dest,
                                      r3dScreenBuffer* src );
        virtual void FinishImpl();

        virtual void PushDefaultSettingsImpl();
        …
        …

        // data
private:
        …
        …

        SettingsArr    mSettingsArr;
};
```

The following methods must be overriden:

```
        virtual void InitImpl();
        virtual void CloseImpl();
        virtual void PrepareImpl(     r3dScreenBuffer* dest,
                                      r3dScreenBuffer* src );
        virtual void FinishImpl();
```

Example of *InitImpl* follows:

```
void
PFX_Copy::InitImpl()
{
        mData.PixelShaderID = r3dRenderer->GetPixelShaderIdx( "PS_COPY" );
}
```

InitImpl is called once when the Post FX is created. Because this post process demands no custom vertex shader or texture transformation, only pixel shader id is filled. The rest of the fields is filled automaticly.

CloseImpl can be left empty in our case, because all shaders are deallocated by the engine automaticly:

```
void
PFX_Copy::CloseImpl()
{
}
```

If the post process allocates any custom resources, they should be freed in its *CloseImpl* imlementation.

PrepareImpl is called each time before the post process is about to be rendered. Here is the example implementation:

```
void
PFX_Copy::PrepareImpl( r3dScreenBuffer* dest, r3dScreenBuffer* src )
{
	r3dRenderer->SetRenderingMode( mBlendMode | R3D_BLEND_PUSH );

	const Settings& sts = mSettingsArr[ 0 ];

	mData.TexTransform[ 0 ] = sts.TexScaleX ;
	mData.TexTransform[ 1 ] = sts.TexScaleY ;

	if( fabs( dest->Width - src->Width * sts.TexScaleX ) > 0.125f
			||
		fabs( dest->Height - src->Height * sts.TexScaleY ) > 0.125f
		)
	{
		r3dSetFiltering( R3D_BILINEAR, 0 );
		mNeedRestoreFiltering = 1 ;
	}
	else
	{
		mNeedRestoreFiltering = 0 ;
	}

	if( mColorWriteMask != PostFXChief::DEFAULT_COLOR_WRITE_MASK )
	{
		D3D_V( r3dRenderer->pd3ddev->SetRenderState( D3DRS_COLORWRITEENABLE,
		                                             mColorWriteMask ) );
	}

	if( sts.SkyOnly )
	{
		SetupLightMaskStencilStates( SCM_UNLITAREA );
		D3D_V( r3dRenderer->pd3ddev->SetRenderState( D3DRS_STENCILENABLE, TRUE ) ) ;
	}
}
```

Note: because PFX_Copy effect supports execution several times per frame, with different post fx configuration, it maintains an arrays of settings. These settings are pushed onto the stack each time an effect is appended. This may be either done by the user explicitely, or automaticly (in case this wasn't done by the user), via *PushDefaultSettingsImpl* call. The user manual configuration is done via function `PushSettings` in our case. This function has the following body:

```
void
PFX_Copy::PushSettings( const Settings& settings )
{
	mSettingsArr.PushBack( settings );

	mSettingsPushed = 1;
}
```

Please note the triggering of *mSettingsPushed* variable. This variable is defined in *PostFX* – our base class. After this variable has been set to 1, no default settings pushing is done by the system. Any user specific settings pushing should be accompanied by setting this variable to non-zero value.

FinishImpl should restore the states, altered by *PrepareImpl* function. Here is our PFX_Copy implementation.

```
void
PFX_Copy::FinishImpl()
{
	r3dRenderer->SetRenderingMode( R3D_BLEND_POP );

	if( mColorWriteMask != PostFXChief::DEFAULT_COLOR_WRITE_MASK )
	{
		g_pPostFXChief->SetDefaultColorWriteMask();
	}

	if( mNeedRestoreFiltering )
	{
		if( g_pPostFXChief->GetZeroTexStageFilter() )
			r3dSetFiltering( R3D_BILINEAR, 0 );
		else
			r3dSetFiltering( R3D_POINT, 0 );

		mNeedRestoreFiltering = 0 ;
	}

	const Settings& sts = mSettingsArr[ 0 ];

	if( sts.SkyOnly )
	{
		D3D_V( r3dRenderer->pd3ddev->SetRenderState( D3DRS_STENCILENABLE, FALSE ) ) ;
	}

	mSettingsArr.Erase( 0 );
}
```

Please note the erasing of the 0 entry of the settings array. Because pushing of the settings is done regardless of the user specifiying, this should be done after each effect execution in order to advance to next settings.

Following is our example imlementation of *PushDefaultSettingsImpl*. It should be implemented only in case the effect supports execution multiple times per frame.

```
void
PFX_Copy::PushDefaultSettingsImpl()
{
	mSettingsArr.PushBack( Settings() ) ;
}
```

Default settings are being setup in *Settings* default constructor.