StubObject::CreateNew copies an object.iff file template to make a newobject.iff file.
Iterates over all resource types, id's, and values, copying them.

SprMakes notes:

When flipping a sprite horizontally, the x bbox changes by 1 because the x origin is between pixels.
If there is a bmp file with the same name as the tga file, it uses the bmp file for the palette.
The z-buffer comes from the tga alpha channel

old 'SPR#' and new 'SPR2' sprite formats. old still used?
old "1.0" and new "2.0" iff file formats. old still used?

Water tiles have alpha in z channel. Palette from bmp, color and alpha from tga.

Resources required to make an object:
	sprites
	catalog description
	catalog icon
	catalog picture

Here are the files required to read and preview sprites:

FileUtils.cpp
FlatFile.cpp
IFFResFile.cpp
IFFResFile2.cpp
PalWrap.cpp
Recon.cpp
ResFile.cpp
RMemWrap.cpp
StringBuffer.cpp
StrSet.cpp
Swizzle.cpp
XAssert.cpp

Secrets to using Skin modifier from MaxScript:

skinOps
#Struct:skinOps(
  GetNumberBones:<fn>,
  GetSelectedBone:<fn>,
  ReplaceVertexWeights:<fn>,
  GetVertexWeightBoneID:<fn>,
  GetVertexWeightCount:<fn>,
  SetOuterRadius:<fn>,
  GetInnerRadius:<fn>,
  SelectBone:<fn>,
  SetVertexWeights:<fn>,
  GetVertexWeight:<fn>,
  GetNumberVertices:<fn>,
  SetInnerRadius:<fn>,
  GetNumberCrossSections:<fn>,
  IsVertexModified:<fn>,
  SelectVertices:<fn>,
  GetBoneName:<fn>,
  IsVertexSelected:<fn>,
  GetOuterRadius:<fn>)

```
skinOps.getnumberbones sk
28

From C:\3dsmax\Maxsdk\SAMPLES\Modifier\BonesDef\CommandModes.cpp


Memory optimizations:
        Automatically flush CFTFile cache's after a while.

Decompressing and rendering sprites:
        inRP->GetZBufferLookup()
        GetZSpriteLookup()
        int zorigin = z_lookup->GetVerticalZValue(inRP,inRP->mCenterOfTileY);

                                        if ( z <= inDestZ[i] ) {                //
note: no bias
                                                inDestPx[i] = Blend555(
inDestPx[i],

inClut[decompPx[flip_i]],

decompAlpha[flip_i] );
                                                inDestZ[i] = UInt16(z);
                                        }

        cAnimDeviceSub::GetLengthOfClipBox(int inScale)

        // zoom is 1,2,3:far, medium, close
        // xorigin,yorigin is pixel offset of tile coord 0,0 altitude 0
        // rotation is 0-3 and is the same as world rotation
        void SetCameraPosition(Int zoom, Int xorigin, Int yorigin, Int rotation);

// scale is related to zoom by:  zoom = 2^(scale-1).  So scale=1 means no zoom,
// scale=2 means zoom by factor of 2, etc.

cAnimDeviceSub::cAnimDeviceSub(void)
: mIsoToWorld(mat4(vec4(0,0,kIsoToWorldConversionFactor,0),
                                vec4(kIsoToWorldConversionFactor,0,0,0),
                                vec4(0,kIsoToWorldConversionFactor,0,0),
                                vec4(0,0,0,1))),

  mWorldToIso(mat4(vec4(0,0,kWorldToIsoConversionFactor,0),
                                vec4(kWorldToIsoConversionFactor,0,0,0),
                                vec4(0,kWorldToIsoConversionFactor,0,0),
                                vec4(0,0,0,1)))

// Tokens are stored as 16 bits; the high 3 bits is the Token,
// the low 13 bits is the Data.
// General notes:
//              * Only clear space is compressed...it's assumed that z/pixel pairs
//                and z/pixel/alpha triplets are not amenable to RLE compression.
//                Presumably some other format decompresses into one of these
streams.
// Token notes:
//              * 1st token is always kNL or kSL
//              * All non-blank lines start with kNL
//              * ZP runs are interleaved [z][px][z][px][z][pz]...
//              * ZPA runs are interleaved [z][px][a][z][px][a]...
//              * kSL(1) is illegal; use kNL (performance reasons)
//              * kSL(0) is also illegal (duh)
//              * skipping 2 lines, 3 lines, etc.:
//                      * line of pixels, blank line, line of pixels ->
//                              kRunXXX...kNL kNL kRunXXX...
//                              * Note that kSL(1) is illegal; kSL(2) is illegal
```

```
in
//                                this context because all non-blank lines start
with kNL.
//                        * line of px, blank line, blank line, line of pixels ->
//                                kRunXXX...kSL(2) kNL kRunXXX
//                        * and so on
// Data notes:
//                * data for kNL is bytes in stream until next kNL, kSL or kEOS from
address
//                  of 16-bit token/data pair
//                * data for kRunZP, kRunZPA, and kClear is # of pixels
//                * data for kSL is # of lines to skip
//                * data for kEOS is undefined


Mac and Linux porting issues:
        __int64 data type not implemented by all compilers
        need gimex and real library source


WaterSpr2Decoder.cpp:
  Modified version of the Spr2Decoders for decoding water tiles.  Water tiles have
  alpha information in their z-channel, and z-information is supplied separately
through
  lookup tables (like the floor tiles)


Rect cRenderer::GetShapeBounds(int listID, int index)
{
        SpriteListInfo* info = GetListInfo(listID);
        if (info) {
                if (info->GetVersion() < kEndOfOldVersions) {
                        Shape *aShape = info->GetPixelShape(index);
                        XASSERT( aShape != NULL, "GetShapeBounds failed");
                        return aShape->rect;
                } else if (info->GetVersion() > kEndOfOldVersions) {
                        Rect aRect;
                        Sprite2 *aSpr2 = info->GetSPR2( index );
                        aRect.left = aRect.top = 0;
                        aRect.right = aSpr2->mWidth;
                        aRect.bottom = aSpr2->mHeight;
                        return aRect;
                } else {
                        XASSERT(false,"unknown version info...");
                        return fBadRect;
                }
        } else {
                return fBadRect;
        }
}


phy = $body.modifiers[1]
pce = getPhyContextExport $body phy
convertToRigid pce true
allowBlending pce true
verts = pce.numVerts
vi = getVertexInterface pce 1
vi.numNodes
```