Updating Plug-ins to support the new Paramblock2 interface

Introduction

With the release of 3ds Max 3 came the introduction of a new Parameter Block and Parameter Map mechanism. The goal for the system was to make it possible for plug-ins to host all of their user-visible parameters in one or more parameter blocks, including complex parameters such as ReferenceTargets, Sub-Anims, dynamic parameter tables (Tabs<>), and class parameters. Further, the new parameter blocks make handling of old-version loading and reference management automatic.

To help support these new parameters the Parameter Map mechanism was updated to provide automatic UI creation for the new parameter types, including common 3ds Max controls such as node pickers, texmap selectors, and list boxes for tabular parameters.

The new parameter block system exposed the plug-in data to systems like the MAXScript, the Macro Recorder and the Schematic View.

Overview of the Main new Classes

ClassDesc2

All plug-ins have a ClassDesc. From 3ds Max R3 a new class descriptor, sub-classed from ClassDesc is used. ClassDesc2 maintains a table of ParamBlockDesc2's and allows for automatic creation of the User Interface.

IParamBlock2

This is the interface into the parameter blocks. It contains all the necessary GetValue() and SetValue() methods needed to access the new forms of data.

ParamBlockDesc2

An instance of this class contains the descriptive data for a ParamBlock2 and all the parameters it contains. These are created using a var-args constructor, the arguments to which define the block and its parameters. A table of all the parameter block descriptors for a plug-in class is kept in its ClassDesc2 instance. A ParamBlockDesc2 contains an array of ParamDef instances, one for each parameter, and a set of block-level flags and UI information. A ParamDef structure contains a list of flags and descriptions for describing the user interface.

ParamBlock2PLCB

Instances of this class are given to ILoad::RegisterPostLoadCallback() to enable automatic conversion of old-parameter block versions of the plug-in. Developers give the constructor for this class a ParamVersionDesc array and the current ParamBlockDesc2 and it will load old parameter block objects into ParamBlock2 objects.

Tutorial 1 – Converting the Bend Modifier

The aim of this tutorial is to explain the steps required in converting an existing plug-in from ParamBlock to ParamBlock2. To aid in this the Bend modifier found in the samples under "maxsdk\howto" will be converted. At each point the key issues will be addressed. The final converted plug-in is contained in the accompanying zip file. The source code should be read in conjunction with this document.

The tutorial is a descriptive step by step guide on how to convert an existing plug-in to support Paramblock2 interface. However the new system has many methods and variables so it is advisable to cross-reference this document with the 3ds Max SDK and the accompanying source code.

Note that at the time the Bend modifier was converted from Paramblock to Paramblock2, the conversion did not support Save To Previous. All plug-ins should always properly support Save to Previous. For an example of doing so, see method 'bool BoxObject::SpecifySaveReferences(ReferenceSaveManager& referenceSaveManager)' in Tutorial 2 – Converting the Box Object.

Step 1 – Project Settings

To use the new methods, the following need to be added to the project:

- Include File - iparamm2.h
  Library file - paramblk2.lib

Step 2 – Convert SimpleMod

The bend Modifier is sub classed from SimpleMod. However there is a new class that adds support for Paramblock2 called SimpleMod2. This has a public data member, IParamBlock2 *pblock2, used instead of the IParamBlock *pblock provided by SimpleMod. It also provides implementations of ReferenceMaker::GetReference() and SetReference() which get and set the pblock2 pointer.

Original interfaces that are now redundant and can be deleted are as follows:-

ParamArray
ParamMaps
GetParamName()
GetParamDim()

Also, the notification messages **REFMSG_GET_PARAM_NAME** and **REFMSG_GET_PARAM_DIM** are no longer required

There are three additions to be made and these add direct access to the Parameter blocks maintained by the plug-in. These methods are from Class Animatable

NumParamBlocks()
IParamBlock2* GetParamBlock(int i)
IParamBlock2* GetParamBlockByID(BlockID id)

The complete definition of BendMod is as follows:-

```
class BendMod : public SimpleMod2 {
	public:

		BendMod();

		// --- Interhited virtual methods of Animatable
		void DeleteThis() { delete this; }
		void GetClassName(TSTR& s) { s= GetString(IDS_RB_BENDMOD); }
		virtual Class_ID ClassID() { return BEND_CID;}

		void BeginEditParams( IObjParam  *ip, ULONG flags,Animatable *prev);
		void EndEditParams( IObjParam *ip,ULONG flags,Animatable *next);

		// Add Direct Paramblock2 Support

		int  NumParamBlocks() { return 1; }
		IParamBlock2* GetParamBlock(int i) { return pblock2; }
		IParamBlock2* GetParamBlockByID(BlockID id) { return (pblock2->ID() == id) ? pblock2 : NULL; }

		// --- Interhited virtual methods of ReferenceMaker
		IOResult Load(ILoad *iload);

		// --- Interhited virtual methods of ReferenceTarget
		RefTargetHandle Clone(RemapDir& remap = NoRemap());

		// --- Interhited virtual methods of BaseObject
		TCHAR *GetObjectName() { return GetString(IDS_RB_BEND2);}

		// --- Interhited virtual methods of SimpleMod
		Deformer& GetDeformer(TimeValue t,ModContext &mc,Matrix3& mat,Matrix3& invmat);
		Interval GetValidity(TimeValue t);

		BOOL GetModLimits(TimeValue t,float &zmin, float &zmax, int &axis);
		void InvalidateUI();
};
```

Step 3 – Build the ParamBlockDesc2

The original modifier created its interface by using ParamUIDesc and ParamBlockDescID. Now it is all controlled by ParamBlockDesc2.   The bend modifier has a relatively simple interface; for a more detailed look at ParamBlockDesc2 please refer to the PB2Utility sample.

First of all a list of parameter Ids are defined for use with the block descriptor and also for use with GetValue and SetValue. The order you declare these in is the order that **they must** be defined in the descriptor.

```
enum { bend_params,};
enum { bend_angle,
	bend_dir,
	bend_axis,
	bend_fromto,
	bend_to,
	bend_from,
};
```

The first line for the ParamBlockDesc2 constructor defines the block:

```
static ParamBlockDesc2 bend_param_blk ( bend_params, _T("Bend Parameters"),  0, &bendDesc, P_AUTO_CONSTRUCT + P_AUTO_UI, SIMPMOD_PBLOCKREF,
	//rollout
	IDD_BENDPARAM, IDS_RB_PARAMETERS, 0, 0, NULL,
```

The flag P_AUTO_CONSTRUCT tells the system that the reference will be created and handled by the owner. If it is set then the reference number must be given after the flags, in this case SIMPMOD_PBLOCKREF. The actual creation is achieved in the call to ClassDesc2::MakeAutoParamBlocks(). This will create the Parameter map and also create the references using the ref number supplied.

If P_AUTO_UI is used, then this tells the system that UI will be automatically created during calls to ClassDesc2::BeginEditParams(). If this is specified then further dialog information is required including Dialog ID, Dialog Name, Append Closed Flag, and the Dialog proc to handle additional initialization (in this case it is set to NULL).

Once the block has been defined, the parameters need to be defined. The definitions for all the parameters are very similar so only one will be described here:

```
bend_to, _T("BendTo"),      TYPE_FLOAT,     P_ANIMATABLE,             IDS_TO,
            p_default,           0.0f,
            p_range,             0.0f,        BIGFLOAT,
            p_ui,                TYPE_SPINNER, EDITTYPE_UNIVERSE, IDC_BEND_TO, IDC_BEND_TOSPIN, SPIN_AUTOSCALE,
            p_accessor,          &bendPBAccessor,
            end,
```

The first line contains required elements. The first two elements are ID and internal name. The next defines the parameter type; in this case it is a float value. A list of flags follows in which P_ANIMATIBLE is defined which means that the parameter can be animated. The last entry is the local name which will be used by Trackview, Schematic View and MAXScript. When assigning the internal and localized names, be careful that the names given are not the same as any of the MAXScript node level property names, such a 'Position'. To see the complete list of MAXScript node level property names type into the MAXScript Listener: 'print (getpropnames node)'. Following this is a list of details, which define how the parameter will work. In particular is P_UI, which specifies what Dialog Resource will control the parameter. p_accessor holds the pointer of a PBAccessor class which allows you to check the data of the parameter during calls to GetValue () and SetValue(). More details on PBAccessor can be found in **Step 4**.

When assigning the non-localized parameter name for each parameter, use the non-localized name in the explicit MAXClass descriptor for the class, if any, in dll\maxscrpt\maxclses.cpp. The p_ms_default values also come from the explicit MAXClass descriptor.

The p_ui IDC_* values are controls in the dialog definition in the resources file, and come from corresponding pb1 code's ParamUIDesc instances.

The p_range values come from corresponding pb1 code's ParamUIDesc instances.

The p_default values typically come from the static initialization values from pb1 version of the class and its ResetClassParams method, it present.

When specifying the type of the parameter, and the pb1 type is TYPE_FLOAT, look at pb1 code's GetParameterDim method to see what dimensioning is used. The following table shows the pb2 parameter type to use for each parameter dimension type.

PB1 Parameter Dimension Type	PB2 Parameter Type
defaultDim	TYPE_FLOAT
stdWorldDim	TYPE_WORLD
stdAngleDim	TYPE_ANGLE
stdColorDim	TYPE_FLOAT
stdColor255Dim	TYPE_COLOR_CHANNEL
stdPercentDim	TYPE_PCNT_FRAC
stdNormalizedDim	TYPE_FLOAT
stdSegmentsDim	TYPE_FLOAT
stdTimeDim	TYPE_TIMEVALUE

The full ParamBlockDesc2 is listed below.

```
static ParamBlockDesc2 bend_param_blk ( bend_params, _T("Bend Parameters"),  0, &bendDesc, P_AUTO_CONSTRUCT + P_AUTO_UI, SIMPMOD_PBLOCKREF,

//Dialog rollout
            IDD_BENDPARAM, IDS_RB_PARAMETERS, 0, 0, NULL,

// params

            bend_angle,                 _T("BendAngle"), TYPE_FLOAT,     P_ANIMATABLE, IDS_ANGLE,
                        p_default,           0.0f,
                        p_range,             -BIGFLOAT, BIGFLOAT,
                        p_ui,                TYPE_SPINNER, EDITTYPE_FLOAT, IDC_ANGLE, IDC_ANGLESPINNER, 0.5f,
                        end,

            bend_dir,                   _T("BendDir"),     TYPE_FLOAT,     P_ANIMATABLE, IDS_DIR,
                        p_default,           0.0f,
                        p_range,             -BIGFLOAT, BIGFLOAT,
                        p_ui,                TYPE_SPINNER, EDITTYPE_FLOAT, IDC_DIR, IDC_DIRSPINNER, 0.5f,
                        end,


            bend_axis,                  _T("bendAxis"), TYPE_BOOL, 0,     IDS_AXIS,
                        p_default,           2,
                        p_ui,                TYPE_RADIO, 3,IDC_X,IDC_Y,IDC_Z,
                        p_vals,              0,1,2,
                        end,
            bend_fromto,               _T("FromTo"), TYPE_BOOL, 0,       IDS_FROMTO,
                        p_default,       FALSE,
                        p_ui,            TYPE_SINGLECHEKBOX, IDC_BEND_AFFECTREGION,
                        end,


            bend_to,                   _T("BendTo"),      TYPE_FLOAT,     P_ANIMATABLE, IDS_TO,
                        p_default,       0.0f,
                        p_range,         0.0f, BIGFLOAT,
                        p_ui,            TYPE_SPINNER, EDITTYPE_UNIVERSE, IDC_BEND_TO, IDC_BEND_TOSPIN, SPIN_AUTOSCALE,
                        p_accessor,      &bendPBAccessor,
                        end,

            bend_from,                 _T("BendFrom"),  TYPE_FLOAT,     P_ANIMATABLE, IDS_FROM,
                        p_default,       0.0f,
                        p_range,         -BIGFLOAT, 0.0f,
                        p_ui,            TYPE_SPINNER, EDITTYPE_UNIVERSE, IDC_BEND_FROM, IDC_BEND_FROMSPIN, SPIN_AUTOSCALE,
                        p_accessor,      &bendPBAccessor,
                        end,


            end
            );
```

If the plug-in that you are converting maintains references that are handled in a non-trivial manner, then it may be easier to specify the parameter is of type P_OWNERS_REF. This means that the owner of the paramblock will handle the references not the block itself. When using the P_OWNERS_REF flag the p_ref specification needs to filled out with the ref number to use. Using this system means that the references will be handled in the same way as in your original plug-in.

Step 4 – Parameter Checking

In the original Bend Modifier a dialog procedure was used to control the values of the "to/From" parameters. With the ParamBlock2 system, a new class called PBAccessor has been implemented to allow developers to have a SetValue and GetValue call back mechanism. This allows continuous monitoring of the parameter change. PBAccessor has two methods, Get() and Set(). In the bend modifier case Set() has been used to check the values of 'bend_from' and 'bend_to'.

PBAccessor::Set() is passed a PB2Value Structure which holds the data being changed and an ID of the parameter changing. The following PBAccessor changes the value of bend_from and bend_to depending on a comparison to each other.

```
class bendPBAccessor : public PBAccessor
{
public:
          void Set(PB2Value& v, ReferenceMaker* owner, ParamID id, int tabIndex, TimeValue t)    // set from v
          {
                    BendMod* u = (BendMod*)owner;
                    IParamMap2* pmap = u->pblock2->GetMap();
                    float from, to;

                    switch(id)
                    {
                              case bend_from:
                                        u->pblock2->GetValue(bend_to,t,to,FOREVER);
                                        from = v.f;
                                        if (from >to) {
                                                  u->pblock2->SetValue(bend_to,t,from);
                                        }
                                        break;
                              case bend_to:
                                        u->pblock2->GetValue(bend_from,t,from,FOREVER);
                                        to = v.f;
                                        if (from>to) {
                                                  u->pblock2->SetValue(bend_from,t,to);
                                        }
                                        break;
                    }
          }
};
```

Step 5 – Loading Old Data

Paramblock2 provides an automatic mechanism for loading in old paramap data. To take advantage of this system the original ParamBlockDesc is used, but now using the newly created parameter Ids.

```
static ParamBlockDescID descVer0[] = {
          { TYPE_FLOAT, NULL, TRUE, bend_angle },
          { TYPE_FLOAT, NULL, TRUE, bend_dir },
          { TYPE_INT, NULL, FALSE, bend_axis } };

// The current version
static ParamBlockDescID descVer1[] = {
          { TYPE_FLOAT, NULL, TRUE, bend_angle },
          { TYPE_FLOAT, NULL, TRUE, bend_dir },
          { TYPE_INT, NULL, FALSE, bend_axis },
          { TYPE_INT, NULL, FALSE, bend_fromto },
          { TYPE_FLOAT, NULL, TRUE, bend_to },
          { TYPE_FLOAT, NULL, TRUE, bend_from } };

static ParamVersionDesc versions[] = {
          ParamVersionDesc(descVer0,3,0)
          };
```

In BendMod::Load() method a callback is registered which maps the incoming IDs to new enumerated IDs.   It is important to note that old Parameter maps still need to be loaded using the original mapping techniques before the new parmblock Ids are used.

```
ParamBlock2PLCB* plcb = new ParamBlock2PLCB(versions, 1, &bend_param_blk, this, SIMPMOD_PBLOCKREF);
iload->RegisterPostLoadCallback(plcb);
```

NOTE: the BendMod does not implement the SpecifySaveReferences method, but it should in order to properly support Save To Previous. See Tutorial 2 for a description of implementing the Load and SpecifySaveReferences methods.

Step 6 – Derive from ClassDesc2

ClassDesc2 is subclassed from ClassDesc and it is used for plug-ins using the Paramblock2 system. It contains a table of ParamBlockDesc2s for all the parameter blocks used by the plug-in and a number of methods including access to the block descriptors, auto user interface management, auto param block2 construction, and access to any automatically-maintained ParamMap2s. To use this class, simply replace all reference of ClassDesc with ClassDesc2.

Step 7 – Rename pblock to pblock2

All GetValue() and SetValue() calls used the original pblock pointer to IParamBlock. This needs to be changed so that it uses the IPramBlock2 pointer pblock2.

Tutorial 2 – Converting the Box Object

The aim of this tutorial is to explain the steps required in converting an existing plug-in from ParamBlock to ParamBlock2. The Box object example found in the **maxsdk\samples** will be converted. The “before” and “after” files can be found in **maxsdk\help\PB1 to PB2 Conversion**

The following table shows the differences when converting the Box classes from ParamBlock to ParamBlock2. The first column is the code using ParamBlock, the second is the code using ParamBlock2, and the third is a description of the changes. Text in red shows code that was changed in the code using ParamBlock, text in green shows code that was changed in the code using ParamBlock2. In addition to these changes, ‘paramblk2.lib’ needs to be added as a library file in the project file.

NOTE: a standardized unit test structure can be used for testing a pb1 to pb2 conversion. See maxsdk\samples\objects\Prim_PB1_to_PB2_conversion.UnitTest.ms for an example unit test.

boxobj_pb1.cpp	boxobj_pb2.cpp	
/*** *< FILE: boxobj_pb1.cpp DESCRIPTION: A Box object implementation using ParamBlock *> Copyright (c) 1994, All Rights Reserved. ***/ #include "prim.h"	/*** *< FILE: boxobj_pb2.cpp DESCRIPTION: A Box object implementation using ParamBlock2 *> Copyright (c) 1994, All Rights Reserved. ***/ #include "prim.h"	
#include "iparamm.h"	#include "iparamm2.h"	Include iparamm2.h instead of iparamm.h.

#include "Simpobj.h"
 #include "surf_api.h"
 #include "MNMath.h"
 #include "PolyObj.h"
 #include "macroRec.h"
 #include "RealWorldMapUtils.h"
	#include "Simpobj.h"
	 #include "surf_api.h"
	 #include "MNMath.h"
	 #include "PolyObj.h"
	 #include "macroRec.h"
	 #include "RealWorldMapUtils.h"	

	#include <ReferenceSaveManager.h>	Include ReferenceSaveManager.h to support saving to previous (pb1-based) versions.

class BoxObject : public GenBoxObject, public IParamArray, public RealWorldMapSizeInterface
 {
	class BoxObject : public GenBoxObject, public RealWorldMapSizeInterface {	No longer derive from IParamArray.

private:
   bool mPolyBoxSmoothingGroupFix;
 	
 public:
   // Class vars
	private:
	   bool mPolyBoxSmoothingGroupFix;
	
	 public:
	   // Class vars	

boxobj_pb1.cpp	boxobj_pb2.cpp	
static IParamMap *pmapCreate; static IParamMap *pmapTypeIn; static IParamMap *pmapParam; static IObjParam *ip; static int dlgLSegs; static int dlgWSegs; static int dlgHSegs; static int createMeth; static Point3 crtPos; static float crtWidth, crtHeight, crtLength;	static IObjParam *ip; static bool typeinCreate;	Remove the static IParamMap* members. Whether to create as a box or a cube is now stored in a ParamBlock2 that is handled as class parameters on the box's ClassDesc2. The type in creation parameters are now stored in a ParamBlock2 that is handled as class parameters on the box's ClassDesc2. The class creation parameters are now accessed via the class's ParamBlock2 instances, so no longer need the static members here that were holding this data. We do need to know when we are creating the box instance via type in create so that we use the class creation parameters.
BoxObject(BOOL loading); // From Object int CanConvertToType(Class_ID obtype); Object* ConvertToType(TimeValue t, Class_ID obtype); void GetCollapseTypes(Tab<Class_ID> &clist, Tab<TSTR*> &nlist); // From BaseObject CreateMouseCallBack* GetCreateMouseCallBack(); void BeginEditParams(IObjParam *ip, ULONG flags, Animatable *prev); void EndEditParams(IObjParam *ip, ULONG flags, Animatable *next); const TCHAR *GetObjectName() { return GetString(IDS_RB_BOX); } BOOL HasUVW(); void SetGenUVW(BOOL sw); // Animatable methods void DeleteThis() { delete this; } Class_ID ClassID() { return Class_ID(BOXOBJ_CLASS_ID, 0); }	BoxObject(BOOL loading); // From Object int CanConvertToType(Class_ID obtype) override; Object* ConvertToType(TimeValue t, Class_ID obtype) override; void GetCollapseTypes(Tab<Class_ID> &clist, Tab<TSTR*> &nlist) override; // From BaseObject CreateMouseCallBack* GetCreateMouseCallBack()override; void BeginEditParams(IObjParam *ip, ULONG flags, Animatable *prev) override; void EndEditParams(IObjParam *ip, ULONG flags, Animatable *next) override; const TCHAR *GetObjectName() override { return GetString(IDS_RB_BOX); } BOOL HasUVW() override; void SetGenUVW(BOOL sw) override; // Animatable methods void DeleteThis() override { delete this; } Class_ID ClassID() override { return Class_ID(BOXOBJ_CLASS_ID, 0); }	
// From ref RefTargetHandle Clone(RemapDir& remap);	// From ref RefTargetHandle Clone(RemapDir& remap) override;	
	bool SpecifySaveReferences(ReferenceSaveManager& referenceSaveManager) override;	Add SpecifySaveReferences method declaration for support of Save to Previous.
IOResult Save(ISave *isave); IOResult Load(ILoad *iload);	IOResult Save(ISave *isave) override; IOResult Load(ILoad *iload) override;	
// From IParamArray BOOL SetValue(int i, TimeValue t, int v); BOOL SetValue(int i, TimeValue t, float v); BOOL SetValue(int i, TimeValue t, Point3 &v); BOOL GetValue(int i, TimeValue t, int &v, Interval &ivalid); BOOL GetValue(int i, TimeValue t, float &v, Interval &ivalid); BOOL GetValue(int i, TimeValue t, Point3 &v, Interval &ivalid);		Delete the IParamArray class's methods. This functionality is now handled by IParamBlock2.
// From SimpleObject void BuildMesh(TimeValue t); BOOL OKtoDisplay(TimeValue t); void InvalidateUI();	// From SimpleObject void BuildMesh(TimeValue t) override; BOOL OKtoDisplay(TimeValue t) override; void InvalidateUI()override;	
ParamDimension *GetParameterDim(int pbIndex); TSTR GetParameterName(int pbIndex);		Delete the GetParameterDim and GetParameterName methods. This functionality is now handled by IParamBlock2.
// From GenBoxObject void SetParams(float width, float height, float length, int wsegs, int lsegs, int hsegs, BOOL genUV); // Get/Set the UsePhyicalScaleUVs flag. BOOL GetUsePhysicalScaleUVs();	// From GenBoxObject void SetParams(float width, float height, float length, int wsegs, int lsegs, int hsegs, BOOL genUV) override; // Get/Set the UsePhyicalScaleUVs flag. BOOL GetUsePhysicalScaleUVs() override;	

boxobj_pb1.cpp	boxobj_pb2.cpp	

void SetUsePhysicalScaleUVs(BOOL flag);
void UpdateUI();
 	
//From FPMixinInterface
BaseInterface* GetInterface(Interface_ID id)
{
if (id == RWS_INTERFACE)
return (RealWorldMapSizeInterface*)this;
 	
BaseInterface* intf = GenBoxObject::GetInterface(id);
if (intf)
return intf;
 	
return FPMixinInterface::GetInterface(id);
}
 	
// local
Object *BuildPolyBox(TimeValue t);
};
 	
// class variables for box class.
IObjParam *BoxObject::ip = NULL;
	void SetUsePhysicalScaleUVs(BOOL flag) override;
	void UpdateUI();
	
	//From FPMixinInterface
	BaseInterface* GetInterface(Interface_ID id) override
	{
	if (id == RWS_INTERFACE)
	return (RealWorldMapSizeInterface*)this;
	
	BaseInterface* intf = GenBoxObject::GetInterface(id);
	if (intf)
	return intf;
	
	return FPMixinInterface::GetInterface(id);
	}
	
	// local
	Object *BuildPolyBox(TimeValue t);
	};
	
	// class variables for box class.
	IObjParam *BoxObject::ip = NULL;	

int BoxObject::dlgLSegs = BDEF_SEGS;
int BoxObject::dlgWSegs = BDEF_SEGS;
int BoxObject::dlgHSegs = BDEF_SEGS;
IParamMap *BoxObject::pmapCreate = NULL;
IParamMap *BoxObject::pmapTypeIn = NULL;
IParamMap *BoxObject::pmapParam = NULL;
Point3 BoxObject::crtPos = Point3(0, 0, 0);
float BoxObject::crtWidth = 0.0f;
float BoxObject::crtHeight = 0.0f;
float BoxObject::crtLength = 0.0f;
int BoxObject::createMeth = 0;
	bool BoxObject::typeinCreate = false;
	
	#define PBLOCK_REF_NO 0
		Remove initialization of static members that no longer exist.
		Add initialization of new static member.
		Define the reference number of the IParamBlock2*. The IParamBlock2* is held by the SimpleObject2 base class as reference 0.

#define BMIN_LENGTH float(0)
#define BMAX_LENGTH float(1.0E30)
#define BMIN_WIDTH float(0)
#define BMAX_WIDTH float(1.0E30)
#define BMIN_HEIGHT float(-1.0E30)
#define BMAX_HEIGHT float(1.0E30)
 	
#define BDEF_DIM float(0)
#define BDEF_SEGS 1
 	
#define MIN_SEGMENTS 1
#define MAX_SEGMENTS 200
 	
// in prim.cpp - The dll instance handle
extern HINSTANCE hInstance;
 	
//--- ClassDescriptor and class vars ---------------------------------
 	
static BOOL sInterfaceAdded = FALSE;
	#define BMIN_LENGTH float(0)
	#define BMAX_LENGTH float(1.0E30)
	#define BMIN_WIDTH float(0)
	#define BMAX_WIDTH float(1.0E30)
	#define BMIN_HEIGHT float(-1.0E30)
	#define BMAX_HEIGHT float(1.0E30)
	
	#define BDEF_DIM float(0)
	#define BDEF_SEGS 1
	
	#define MIN_SEGMENTS 1
	#define MAX_SEGMENTS 200
	
	// in prim.cpp - The dll instance handle
	extern HINSTANCE hInstance;
	
	//--- ClassDescriptor and class vars ---------------------------------
	
	static BOOL sInterfaceAdded = FALSE;	

class BoxObjClassDesc :public ClassDesc {	class BoxObjClassDesc :public ClassDesc2 {	Derive class from ClassDesc2 rather than ClassDesc.

public:
int IsPublic() { return 1; }
void * Create(BOOL loading = FALSE)
{
if (!sInterfaceAdded) {
AddInterface(&gRealWorldMapSizeDesc);
sInterfaceAdded = TRUE;
}
return new BoxObject(loading);
}
const TCHAR * ClassName() { return GetString(IDS_RB_BOX_CLASS); }
	public:
	int IsPublic() override { return 1; }
	void * Create(BOOL loading = FALSE) override
	{
	if (!sInterfaceAdded) {
	AddInterface(&gRealWorldMapSizeDesc);
	sInterfaceAdded = TRUE;
	}
	return new BoxObject(loading);
	}
	const TCHAR * ClassName() override { return GetString(IDS_RB_BOX_CLASS); }	

boxobj_pb1.cpp	boxobj_pb2.cpp	

SClass_ID SuperClassID() { return GEOMOBJECT_CLASS_ID; }
Class_ID ClassID() { return Class_ID(BOXOBJ_CLASS_ID, 0); }
const TCHAR* Category() { return GetString(IDS_RB_PRIMITIVES); }
	SClass_ID SuperClassID() override { return GEOMOBJECT_CLASS_ID; }
	Class_ID ClassID() override { return Class_ID(BOXOBJ_CLASS_ID, 0); }
	const TCHAR* Category() override { return GetString(IDS_RB_PRIMITIVES); }	

void ResetClassParams(BOOL fileReset);	const TCHAR* InternalName() { return _T("Box"); } // returns fixed parsable name (scripter-visible name) HINSTANCE HInstance() { return hInstance; } // returns owning module handle	Remove method ResetClassParams. Add methods InternalName and HInstance.

};
 	
static BoxObjClassDesc boxObjDesc;
 	
ClassDesc* GetBoxobjDesc() { return &boxObjDesc; }
	};
	
	static BoxObjClassDesc boxObjDesc;
	
	ClassDesc* GetBoxobjDesc() { return &boxObjDesc; }	

void BoxObjClassDesc::ResetClassParams(BOOL fileReset)
{
BoxObject::dlgLSegs = BDEF_SEGS;
BoxObject::dlgWSegs = BDEF_SEGS;
BoxObject::dlgHSegs = BDEF_SEGS;
BoxObject::crtWidth = 0.0f;
BoxObject::crtHeight = 0.0f;
BoxObject::crtLength = 0.0f;
BoxObject::createMeth = 0;
BoxObject::crtPos = Point3(0, 0, 0);
}
		Remove the ResetClassParams implementation.

//--- Parameter map/block descriptors -------------------------------
 	
// Parameter block indices
#define PB_LENGTH 0
#define PB_WIDTH 1
 	
#define PB_HEIGHT 2
 	
#define PB_WSEGS 3
#define PB_LSEGS 4
#define PB_HSEGS 5
#define PB_GENUVS 6
 	
// Non-parameter block indices
#define PB_CREATEMETHOD 0
#define PB_TI_POS   1
#define PB_TI_LENGTH 2
#define PB_TI_WIDTH   3
#define PB_TI_HEIGHT 4
	// ParamBlockDesc2 IDs
	enum paramblockdesc2_ids { box_creation_type, box_type_in, box_params, };
	enum box_creation_type_param_ids { box_create_meth, };
	enum box_type_in_param_ids { box_ti_pos, box_ti_length, box_ti_width, box_ti_height, };
	enum box_param_param_ids { box_length = BOXOBJ_LENGTH, box_width = BOXOBJ_WIDTH, box_height = BOXOBJ_HEIGHT,
	box_wsegs = BOXOBJ_WSEGS, box_lsegs = BOXOBJ_LSEGS, box_hsegs = BOXOBJ_HSEGS, box_mapping = BOXOBJ_GENUVS, };
		Define enums associated with the object and class ParamBlockDesc2s.
		The class holds three ParamBlockDesc2s with block IDs of box_creation_type, box_type_in, and box_params.
		The ParamBlockDesc2 with block ID box_creation_type defines 1 parameter.
		The ParamBlockDesc2 with block ID box_type_in defines 4 parameters.
		The ParamBlockDesc2 with block ID box_type_in defines 7 parameters. This ParamBlockDesc2 corresponds to the ParamBlockDescID[] descVer1 defined in pb1 code. The parameter ids must match the ids in descVer1 to properly load legacy files.

boxobj_pb1.cpp	boxobj_pb2.cpp	

//
// Creation method
 	
static int createMethIDs[] = { IDC_CREATEBOX,IDC_CREATECUBE };
 	
static ParamUIDesc descCreate[] = {
  // Box/Cube
  ParamUIDesc(PB_CREATEMETHOD,TYPE_RADIO,createMethIDs,2)
};
#define CREATEDESC_LENGH 1
	namespace
	{
	  class CreationType_Accessor : public PBAccessor
	  {
	    void Set(PB2Value& v, ReferenceMaker* owner, ParamID id, int tabIndex, TimeValue t);
	  };
	
	  static CreationType_Accessor creationType_Accessor;
	}
	
	// class creation type block
	static ParamBlockDesc2 box_crtype_blk(**box_creation_type**, _T("BoxCreationType"), 0, &boxObjDesc, P_CLASS_PARAMS + P_AUTO_UI,
	  //rollout
	  IDD_BOXPARAM1, **IDS_RB_CREATIONMETHOD**, **BEGIN_EDIT_CREATE**, 0, NULL,
	  // params
	  box_create_meth, _T("typeinCreationMethod"), TYPE_INT, 0, IDS_RB_CREATIONMETHOD,
	  p_default, 0,
	  p_range, 0, 1,
	  p_ui, TYPE_RADIO, 2, **IDC_CREATEBOX**, **IDC_CREATECUBE**,
	  p_accessor, &**creationType_Accessor**,
	  p_end,
	  p_end
	  );
		Remove ParamUIDescs
		The added CreationType_Accessor is used to Enable/Disable the width and length spinners in the Keyboard Entry rollout, depending on whether creating a box or a cube, as specified by the **box_create_meth** parameter. In the ParamBlockDesc2 definition:
		box_creation_type comes from the Block ID enum
		IDD_BOXPARAM1 and **IDS_RB_CREATIONMETHOD** comes from the CreateCPParamMap call in the pb1 code
		BEGIN_EDIT_CREATE says to display the rollout only in the create panel **box_create_meth** comes from the box_creation_type param IDs enum
		IDC_CREATEBOX and **IDC_CREATECUBE** are controls in the dialog definition in the resources file, and come from **createMethIDs[]** in the pb1 code.
		Specify to use the accessor **creationType_Accessor**

//
// Type in
 	
static ParamUIDesc descTypeIn[] = {
 	
  // Position
  ParamUIDesc(
    PB_TI_POS,
    EDITTYPE_UNIVERSE,
    IDC_TI_POSX,IDC_TI_POSXSPIN,
 	
    IDC_TI_POSY,IDC_TI_POSYSPIN,
    IDC_TI_POSZ,IDC_TI_POSZSPIN,
    float(-1.0E30),float(1.0E30),
    SPIN_AUTOSCALE),
 	
  // Length
  ParamUIDesc(
    PB_TI_LENGTH,
    EDITTYPE_UNIVERSE,
    IDC_LENGTHEDIT,IDC_LENSPINNER,
 	
    BMIN_LENGTH,BMAX_LENGTH,
    SPIN_AUTOSCALE),
 	
  // Width
  ParamUIDesc(
 	
    PB_TI_WIDTH,
    EDITTYPE_UNIVERSE,
    IDC_WIDTHEDIT,IDC_WIDTHSPINNER,
    BMIN_WIDTH,BMAX_WIDTH,
    SPIN_AUTOSCALE),
 	
  // Height
  ParamUIDesc(
    PB_TI_HEIGHT,
    EDITTYPE_UNIVERSE,
	// class type-in block
	static ParamBlockDesc2 box_typein_blk(**box_type_in**, _T("BoxTypeIn"), 0, &boxObjDesc, P_CLASS_PARAMS + P_AUTO_UI,
	  //rollout
	  IDD_BOXPARAM3, **IDS_RB_KEYBOARDENTRY**, BEGIN_EDIT_CREATE, APPENDROLL_CLOSED, NULL,
	  // params
	  box_ti_pos, _T("typeInPos"), TYPE_POINT3, 0, IDS_RB_TYPEIN_POS,
	  p_default, Point3(0, 0, 0),
	  p_range, float(-1.0E30), float(1.0E30),
	  p_ui, TYPE_SPINNER, EDITTYPE_UNIVERSE, IDC_TI_POSX, IDC_TI_POSXSPIN, IDC_TI_POSY, IDC_TI_POSYSPIN, IDC_TI_POSZ, IDC_TI_POSZSPIN, SPIN_AUTOSCALE,
	  p_end,
	  box_ti_length, _T("typeInLength"), TYPE_FLOAT, 0, IDS_RB_LENGTH,
	  p_default, BDEF_DIM,
	  p_range, BMIN_LENGTH, BMAX_LENGTH,
	  p_ui, TYPE_SPINNER, EDITTYPE_UNIVERSE, IDC_LENGTHEDIT, IDC_LENSPINNER, SPIN_AUTOSCALE,
	  p_end,
	  box_ti_width, _T("typeInWidth"), TYPE_FLOAT, 0, IDS_RB_WIDTH,
	  p_default, BDEF_DIM,
	  p_range, BMIN_WIDTH, BMAX_WIDTH,
	  p_ui, TYPE_SPINNER, EDITTYPE_UNIVERSE, IDC_WIDTHEDIT, IDC_WIDTHSPINNER, SPIN_AUTOSCALE,
	  p_end,
	  box_ti_height, _T("typeInHeight"), TYPE_FLOAT, 0, IDS_RB_HEIGHT,
	  p_default, BDEF_DIM,
	  p_range, BMIN_HEIGHT, BMAX_HEIGHT,
	  p_ui, TYPE_SPINNER, EDITTYPE_UNIVERSE, IDC_HEIGHTEDIT, IDC_HEIGHTSPINNER, SPIN_AUTOSCALE,
	  p_end,
	  p_end
	  );
		Remove ParamUIDescs
		In the ParamBlockDesc2 definition:
		box_type_in comes from the Block ID enum
		IDD_BOXPARAM3 and **IDS_RB_KEYBOARDENTRY** comes from the CreateCPParamMap() call in the pb1 code
		BEGIN_EDIT_CREATE says to display the rollout only in the create panel (from BeginEditParams() in the pb1 code)
		The parameter ids come from the **box_type_in** param IDs enum
		The p_ui IDC_* values are controls in the dialog definition in the resources file, and come from corresponding pb1 code's ParamUIDesc instances.
		The p_range values come from corresponding pb1 code's ParamUIDesc instances.
		The p_default values come from the BoxObject's static initialization values from pb1 code and its ResetClassParams() method.
		The IDS_* values are existing values in the resource file that roughly equivalent to the RTEXT string values specified in the **IDD_BOXPARAM3** dialog definition in the resource file.

boxobj_pb1.cpp

```
    IDC_HEIGHTEDIT,IDC_HEIGHTSPINNER,
    BMIN_HEIGHT,BMAX_HEIGHT,
    SPIN_AUTOSCALE),
};
#define TYPEINDESC_LENGH 4
```

```
//
// Parameters
static ParamUIDesc descParam[] = {
  // Length
  ParamUIDesc(
    PB_LENGTH,
    EDITTYPE_UNIVERSE,
    IDC_LENGTHEDIT,IDC_LENSPINNER,
    BMIN_LENGTH,BMAX_LENGTH,
    SPIN_AUTOSCALE),

  // Width
  ParamUIDesc(
    PB_WIDTH,
    EDITTYPE_UNIVERSE,
    IDC_WIDTHEDIT,IDC_WIDTHSPINNER,
    BMIN_WIDTH,BMAX_WIDTH,
    SPIN_AUTOSCALE),

  // Height
  ParamUIDesc(
    PB_HEIGHT,
    EDITTYPE_UNIVERSE,
    IDC_HEIGHTEDIT,IDC_HEIGHTSPINNER,
    BMIN_HEIGHT,BMAX_HEIGHT,
    SPIN_AUTOSCALE),


  // Length Segments
  ParamUIDesc(
    PB_LSEGS,
    EDITTYPE_INT,
    IDC_LSEGS,IDC_LSEGSPIN,
    (float)MIN_SEGMENTS,(float)MAX_SEGMENTS,
    0.1f),

  // Width Segments
  ParamUIDesc(
    PB_WSEGS,
    EDITTYPE_INT,
    IDC_WSEGS,IDC_WSEGSPIN,
    (float)MIN_SEGMENTS,(float)MAX_SEGMENTS,
    0.1f),

  // Height Segments
  ParamUIDesc(
    PB_HSEGS,
    EDITTYPE_INT,
    IDC_HSEGS,IDC_HSEGSPIN,
    (float)MIN_SEGMENTS,(float)MAX_SEGMENTS,
    0.1f),

  // Gen UVs
  ParamUIDesc(PB_GENUVS,TYPE_SINGLECHECKBOX,IDC_GENTEXTURE),
};
#define PARAMDESC_LENGH 7
```

boxobj_pb2.cpp

```
// per instance box block
static ParamBlockDesc2 box_param_blk(box_params, _T("BoxParameters"), 0, &boxObjDesc,
P_AUTO_CONSTRUCT + P_AUTO_UI, PBLOCK_REF_NO,
  //rollout
  IDD_BOXPARAM2, IDS_RB_PARAMETERS, 0, 0, NULL,
  // params
  box_length, _T("length"), TYPE_WORLD, P_ANIMATABLE + P_RESET_DEFAULT,
IDS_RB_LENGTH,
  p_default, BDEF_DIM,
  p_ms_default, 25.0,
  p_range, BMIN_LENGTH, BMAX_LENGTH,
  p_ui, TYPE_SPINNER, EDITTYPE_UNIVERSE, IDC_LENGTHEDIT, IDC_LENSPINNER,
SPIN_AUTOSCALE,
  p_end,
  box_width, _T("width"), TYPE_WORLD, P_ANIMATABLE + P_RESET_DEFAULT,
IDS_RB_WIDTH,
  p_default, BDEF_DIM,
  p_ms_default, 25.0,
  p_range, BMIN_WIDTH, BMAX_WIDTH,
  p_ui, TYPE_SPINNER, EDITTYPE_UNIVERSE, IDC_WIDTHEDIT, IDC_WIDTHSPINNER,
SPIN_AUTOSCALE,
  p_end,
  box_height, _T("height"), TYPE_WORLD, P_ANIMATABLE + P_RESET_DEFAULT,
IDS_RB_HEIGHT,
  p_default, BDEF_DIM,
  p_ms_default, 25.0,
  p_range, BMIN_HEIGHT, BMAX_HEIGHT,
  p_ui, TYPE_SPINNER, EDITTYPE_UNIVERSE, IDC_HEIGHTEDIT, IDC_HEIGHTSPINNER,
SPIN_AUTOSCALE,
  p_end,
  box_wsegs, _T("widthsegs"), TYPE_INT, P_ANIMATABLE, IDS_RB_WSEGS,
  p_default, BDEF_SEGS,
  p_range, MIN_SEGMENTS, MAX_SEGMENTS,
  p_ui, TYPE_SPINNER, EDITTYPE_INT, IDC_WSEGS, IDC_WSEGSPIN, 0.1f,
  p_end,
  box_lsegs, _T("lengthsegs"), TYPE_INT, P_ANIMATABLE, IDS_RB_LSEGS,
  p_default, BDEF_SEGS,
  p_range, MIN_SEGMENTS, MAX_SEGMENTS,
  p_ui, TYPE_SPINNER, EDITTYPE_INT, IDC_LSEGS, IDC_LSEGSPIN, 0.1f,
  p_end,
  box_hsegs, _T("heightsegs"), TYPE_INT, P_ANIMATABLE, IDS_RB_HSEGS,
  p_default, BDEF_SEGS,
  p_range, MIN_SEGMENTS, MAX_SEGMENTS,
  p_ui, TYPE_SPINNER, EDITTYPE_INT, IDC_HSEGS, IDC_HSEGSPIN, 0.1f,
  p_end,
  box_mapping, _T("mapCoords"), TYPE_BOOL, 0, IDS_RB_GENTEXCOORDS,
  p_default, TRUE,
  p_ms_default, FALSE,
  p_ui, TYPE_SINGLECHECKBOX, IDC_GENTEXTURE,
  p_end,
  p_end
  );
```

Remove ParamUIDescs
In the ParamBlockDesc2 definition:
box_params comes from the Block ID enum
IDD_BOXPARAM2and **IDS_RB_PARAMETERS** comes from the CreateCPParamMap call in the pb1 code
The parameter ids come from the **box_param** param IDs enum
The non-localized parameter name for each parameter comes from the explicit MAXClass descriptor for the class, if any, in dll\maxscrpt\maxclses.cpp.
The parameter type is determined by looking at the corresponding ParamBlockDescID value in ParamBlockDescID[] descVer1 defined in pb1 code, and the ParamDimension returned from the GetParameterDim method for that parameter.
Whether or not a parameter is animatable depends on the 'animatable' member variable for the corresponding ParamBlockDescID value in ParamBlockDescID[] descVer1 defined in pb1 code.
The p_ui IDC_* values are controls in the dialog definition in the resources file, and come from corresponding pb1 code's ParamUIDesc instances.
The p_range values come from corresponding pb1 code's ParamUIDesc instances.
The p_default values come from the BoxObject's static initialization values from pb1 code (float BoxObject::crtWidth etc) and its ResetClassParams method.
The p_ms_default values come from the explicit MAXClass descriptor for the class, if any, in dll\maxscrpt\maxclses.cpp.
The IDS_* values are existing values in the resource file that roughly equivalent to the RTEXT string values specified in the **IDD_BOXPARAM2** dialog definition in the resource file.

boxobj_pb1.cpp	boxobj_pb2.cpp	

	void CreationType_Accessor::Set(PB2Value& v, ReferenceMaker* owner, ParamID id, int tabIndex, TimeValue t)
	{
	  // disable Keyboard Entry Width/Height spinners if creating cube
	  IParamMap2* pmap = boxObjDesc.GetParamMap(&box_typein_blk);
	  if (pmap)
	  {
	    bool createCube = v.i == 1;
	    pmap->Enable(box_ti_width, !createCube);
	    pmap->Enable(box_ti_height, !createCube);
	  }
	}
		Added to control the spinner behavior in the Keyboard Entry rollout base on whether Creation Method is box or cube. If cube, the width and height spinners are disabled.

ParamBlockDescID descVer0[] = {
  { TYPE_FLOAT, NULL, TRUE, PB_LENGTH },
  { TYPE_FLOAT, NULL, TRUE, PB_WIDTH },
  { TYPE_FLOAT, NULL, TRUE, PB_HEIGHT },
  { TYPE_INT, NULL, TRUE, PB_WSEGS },
  { TYPE_INT, NULL, TRUE, PB_LSEGS },
  { TYPE_INT, NULL, TRUE, PB_HSEGS }
};
 	
ParamBlockDescID descVer1[] = {
  { TYPE_FLOAT, NULL, TRUE, PB_LENGTH },
  { TYPE_FLOAT, NULL, TRUE, PB_WIDTH },
  { TYPE_FLOAT, NULL, TRUE, PB_HEIGHT },
  { TYPE_INT, NULL, TRUE, PB_WSEGS },
  { TYPE_INT, NULL, TRUE, PB_LSEGS },
  { TYPE_INT, NULL, TRUE, PB_HSEGS },
  { TYPE_INT, NULL, FALSE, PB_GENUVS }
};
 	
#define **PBLOCK_LENGTH** 7
 	
// Array of old versions
static ParamVersionDesc **versions**[] = {
  ParamVersionDesc(descVer0,6,0),
 	
};
#define **NUM_OLDVERSIONS** 1
 	
#define **CURRENT_VERSION** 1
static ParamVersionDesc **curVersion**(descVer1, PBLOCK_LENGTH, CURRENT_VERSION);
	//--- Parameter map/block descriptors -------------------------------
	
	ParamBlockDescID descVer0[] = {
	  { TYPE_FLOAT, NULL, TRUE, box_length },
	  { TYPE_FLOAT, NULL, TRUE, box_width },
	  { TYPE_FLOAT, NULL, TRUE, box_height },
	  { TYPE_INT, NULL, TRUE, box_wsegs },
	  { TYPE_INT, NULL, TRUE, box_lsegs },
	  { TYPE_INT, NULL, TRUE, box_hsegs }
	};
	
	ParamBlockDescID descVer1[] = {
	  { TYPE_FLOAT, NULL, TRUE, box_length },
	  { TYPE_FLOAT, NULL, TRUE, box_width },
	  { TYPE_FLOAT, NULL, TRUE, box_height },
	  { TYPE_INT, NULL, TRUE, box_wsegs },
	  { TYPE_INT, NULL, TRUE, box_lsegs },
	  { TYPE_INT, NULL, TRUE, box_hsegs },
	  { TYPE_INT, NULL, FALSE, box_mapping }
	};
	
	// Array of old versions
	static ParamVersionDesc **versions**[] = {
	  ParamVersionDesc(descVer0,6,0),
	  ParamVersionDesc(descVer1,7,1),
	};
	#define **NUM_OLDVERSIONS** 2
	
	// ParamBlock data for SaveToPrevious support
	#define **PBLOCK_LENGTH** 7
	#define **CURRENT_VERSION** 1
		The pb1 ParamBlockDescID arrays are still needed in order to support loading of legacy files, and to support Save To Previous.
		The parameter id values must not change in value. By updating to the enum values, you ensure the pb and pb2 parameter id values are the same.
		The old **curVersion** ParamBlockDescID array now becomes an old version placed in the **versions** array, and update **NUM_OLDVERSIONS**
		Still need **PBLOCK_LENGTH** and **CURRENT_VERSION** for Save To Previous
		
		Later Note: The ParamBlockDescID struct has been extended to include an optional 6th member - pb2_id. This is an ID used to identify this parameter in a ParamBlockDesc2.
		There are 2 special values for this member:
		-1 - This ParamBlockDesc parameter is not used in the ParamBlockDesc2
		-2 - Use the 'id' member value as the ParamBlockDesc2 id.
		Otherwise, the value specifies the ParamBlockDesc2 parameter id that corresponds to this parameter.
		If this member is not specified in the initialization list, it defaults to -2.
		This member is used by function CopyParamBlock2ToParamBlock and ParamBlock2PLCB when copying parameters between IParamBlock and IParamBlock2 instances.
		With this change, it is recommended that the original param ID values remain as they were (i.e., use PB_LENGTH, PB_WIDTH, etc.). These param ID values are only used when saving and loading IParamBlock instances, and must not change. These param ID values may or may not correspond to the published Parameter Block IDs (in this case the BOXOBJ_LENGTH, BOXOBJ_WIDTH, etc. values in maxsdk\include\istdplug.h. These values are actually indices into the IparamBlock's params, which correspond the the physical order of ParamBlockDescID instances here.
		It may be simplest to always specify the pb2_id in the ParamBlockDescID initializers just to be certain the mapping is handled correctly.

//--- TypeInDlgProc --------------------------------	//--- TypeInDlgProc --------------------------------	
class BoxTypeInDlgProc : public **ParamMapUserDlgProc** {	class BoxTypeInDlgProc : public **ParamMap2UserDlgProc** {	Derive from **ParamMap2UserDlgProc**
public: BoxObject *ob;	public: BoxObject *ob;	

boxobj_pb1.cpp	boxobj_pb2.cpp	

BoxTypeInDlgProc(BoxObject *o) { ob = o; }
INT_PTR DlgProc(TimeValue t, IParamMap *map, HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);
void DeleteThis() { delete this; }
};
 	
INT_PTR BoxTypeInDlgProc::DlgProc(
TimeValue t, IParamMap *map, HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
switch (msg) {
	BoxTypeInDlgProc(BoxObject *o) { ob = o; }
	INT_PTR DlgProc(TimeValue t, IParamMap2 *map, HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);
	void DeleteThis() { delete this; }
	};
	
	INT_PTR BoxTypeInDlgProc::DlgProc(
	TimeValue t, IParamMap2 *map, HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
	{
	switch (msg)
	{	

case CC_SPINNER_CHANGE:
switch (LOWORD(wParam)) {
case IDC_LENSPINNER:
case IDC_WIDTHSPINNER:
case IDC_HEIGHTSPINNER:
if (ob->createMeth) {
ISpinnerControl *spin = (ISpinnerControl*)lParam;
ob->crtLength = ob->crtWidth = ob->crtHeight =
spin->GetFVal();
map->Invalidate();
}
break;
}
break;
		Code is no longer needed.
		The type in parameter values are now stored in ParamBlockDesc2 box_typein_blk.
		The width and height spinners are disabled when in 'cube' create mode.
		Only the length type in value is used when creating the box instance when in 'cube' create mode. See the pb2 code following `if(createCube)` (two instances).

	case WM_INITDIALOG:
	{
	// disable width and height spinners if in Cube creation mode.
	bool createCube = box_crtype_blk.GetInt(box_create_meth) == 1;
	map->Enable(box_ti_width, !createCube);
	map->Enable(box_ti_height, !createCube);
	}
	break;
		When initializing the rollout's dialog, if in 'cube' create mode disable the width and height spinners.

case WM_COMMAND:
switch (LOWORD(wParam)) {
case IDC_TI_CREATE: {
// We only want to set the value if the object is not in the scene.
	case WM_COMMAND:
	switch (LOWORD(wParam)) {
	case IDC_TI_CREATE: {
	// We only want to set the value if the object is not in the scene.	

if (**ob**->TestAFlag(A_OBJ_CREATING)) {
ob->pblock->SetValue(PB_LENGTH, 0, ob->crtLength);
ob->pblock->SetValue(PB_WIDTH, 0, ob->crtWidth);
ob->pblock->SetValue(PB_HEIGHT, 0, ob->crtHeight);
}
	if (**ob**->TestAFlag(A_OBJ_CREATING)) {
	bool createCube = **box_crtype_blk**.GetInt(**box_create_meth**) == 1;
	if (createCube)
	{
	float val = box_typein_blk.GetFloat(**box_ti_length**);
	ob->pblock2->SetValue(box_length, 0, val);
	ob->pblock2->SetValue(box_width, 0, val);
	ob->pblock2->SetValue(box_height, 0, val);
	}
	else
	{
	ob->pblock2->SetValue(box_length, 0, box_typein_blk.GetFloat(box_ti_length));
	ob->pblock2->SetValue(box_width, 0, box_typein_blk.GetFloat(box_ti_width));
	ob->pblock2->SetValue(box_height, 0, box_typein_blk.GetFloat(box_ti_height));
	}
	}
	else
	BoxObject::typeinCreate = true;
		If have just pushed the Box button, 3ds max has created an instance of the box object, but has not created a node and placed it in the scene. If this case, we operate directly on the box object (**ob**).
		If in 'cube' create mode, determined by getting parameter **box_create_meth** from ParamBlockDesc2 **box_crtype_blk**, get the **box_ti_length** parameter from ParamBlockDesc2 **box_typein_blk** and set the box's **pblock2** parameters **box_length**, **box_width**, and **box_height** to that value.
		If not in 'cube' create mode, set the box's **pblock2** parameters from the corresponding values in **box_typein_blk**.
		If have already created a Box node, then set static member **BoxObject::typeinCreate** to true. The call to NonMouseCreate will cause a new BoxObject instance to be created, and BeginEditParams will be called on that instance. In that method, the type-in parameters will set on the instance if this flag is set.

Matrix3 tm(1);	Matrix3 tm(1);	
tm.SetTrans(ob->crtPos);	tm.SetTrans(**box_typein_blk**.GetPoint3(box_ti_pos));	Get type-in position from ParamBlockDesc2 **box_typein_blk**

boxobj_pb1.cpp	boxobj_pb2.cpp	

ob->suspendSnap = FALSE;
ob->ip->NonMouseCreate(tm);
// NOTE that calling NonMouseCreate will cause this
// object to be deleted. DO NOT DO ANYTHING BUT RETURN.
return TRUE;
}
}
break;
 	
}
return FALSE;
}
	ob->suspendSnap = FALSE;
	ob->ip->NonMouseCreate(tm);
	// NOTE that calling NonMouseCreate will cause this
	// object to be deleted. DO NOT DO ANYTHING BUT RETURN.
	return TRUE;
	}
	}
	break;
	
	}
	return FALSE;
	}	

class BoxParamDlgProc : public **ParamMapUserDlgProc** {	class BoxParamDlgProc : public **ParamMap2UserDlgProc** {	Derive from **ParamMap2UserDlgProc**

public:
BoxObject *mpBoxObj;
HWND mhWnd;
BoxParamDlgProc(BoxObject *o) { mpBoxObj = o; mhWnd = NULL; }
	public:
	BoxObject *mpBoxObj;
	HWND mhWnd;
	BoxParamDlgProc(BoxObject *o) { mpBoxObj = o; mhWnd = NULL; }	

INT_PTR DlgProc(TimeValue t, **IParamMap** *map, HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);	INT_PTR DlgProc(TimeValue t, **IParamMap2** *map, HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);	Argument type **IParamMap2***

void DeleteThis() { delete this; }
void UpdateUI();
BOOL GetRWSState();
};
 	
void BoxParamDlgProc::UpdateUI()
{
if (mhWnd == NULL)
return;
BOOL usePhysUVs = mpBoxObj->GetUsePhysicalScaleUVs();
CheckDlgButton(mhWnd, IDC_REAL_WORLD_MAP_SIZE, usePhysUVs);
EnableWindow(GetDlgItem(mhWnd, IDC_REAL_WORLD_MAP_SIZE), mpBoxObj->HasUVW());
}
 	
BOOL BoxParamDlgProc::GetRWSState()
{
BOOL check = IsDlgButtonChecked(mhWnd, IDC_REAL_WORLD_MAP_SIZE);
return check;
}
 	
INT_PTR BoxParamDlgProc::DlgProc(
	void DeleteThis() { delete this; }
	void UpdateUI();
	BOOL GetRWSState();
	};
	
	void BoxParamDlgProc::UpdateUI()
	{
	if (mhWnd == NULL)
	return;
	BOOL usePhysUVs = mpBoxObj->GetUsePhysicalScaleUVs();
	CheckDlgButton(mhWnd, IDC_REAL_WORLD_MAP_SIZE, usePhysUVs);
	EnableWindow(GetDlgItem(mhWnd, IDC_REAL_WORLD_MAP_SIZE), mpBoxObj->HasUVW());
	}
	
	BOOL BoxParamDlgProc::GetRWSState()
	{
	BOOL check = IsDlgButtonChecked(mhWnd, IDC_REAL_WORLD_MAP_SIZE);
	return check;
	}
	
	INT_PTR BoxParamDlgProc::DlgProc(	

TimeValue t, **IParamMap** *map, HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)	TimeValue t, **IParamMap2** *map, HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)	Argument type **IParamMap2***

{
switch (msg) {
case WM_INITDIALOG: {
mhWnd = hWnd;
UpdateUI();
break;
}
case WM_COMMAND:
switch (LOWORD(wParam)) {
case IDC_GENTEXTURE:
UpdateUI();
break;
 	
case IDC_REAL_WORLD_MAP_SIZE: {
BOOL check = IsDlgButtonChecked(hWnd, IDC_REAL_WORLD_MAP_SIZE);
theHold.Begin();
mpBoxObj->SetUsePhysicalScaleUVs(check);
theHold.Accept(GetString(IDS_DS_PARAMCHG));
mpBoxObj->ip->RedrawViews(mpBoxObj->ip->GetTime());
break;
}
	{
	switch (msg) {
	case WM_INITDIALOG: {
	mhWnd = hWnd;
	UpdateUI();
	break;
	}
	case WM_COMMAND:
	switch (LOWORD(wParam)) {
	case IDC_GENTEXTURE:
	UpdateUI();
	break;
	
	case IDC_REAL_WORLD_MAP_SIZE: {
	BOOL check = IsDlgButtonChecked(hWnd, IDC_REAL_WORLD_MAP_SIZE);
	theHold.Begin();
	mpBoxObj->SetUsePhysicalScaleUVs(check);
	theHold.Accept(GetString(IDS_DS_PARAMCHG));
	mpBoxObj->ip->RedrawViews(mpBoxObj->ip->GetTime());
	break;
	}	

boxobj_pb1.cpp	boxobj_pb2.cpp	

}
    break;
 	
  }
  return FALSE;
}
 	
//--- Box methods -------------------------------
 	
BoxObject::BoxObject(BOOL loading) : mPolyBoxSmoothingGroupFix(true)
{
	}
	    break;
	
	  }
	  return FALSE;
	}
	
	//--- Box methods -------------------------------
	
	BoxObject::BoxObject(BOOL loading) : mPolyBoxSmoothingGroupFix(true)
	{	

ReplaceReference(0, **CreateParameterBlock**(descVer1, PBLOCK_LENGTH, CURRENT_VERSION));
 	
  pblock->SetValue(PB_LSEGS, 0, dlgLSegs);
  pblock->SetValue(PB_WSEGS, 0, dlgWSegs);
  pblock->SetValue(PB_HSEGS, 0, dlgHSegs);
  pblock->SetValue(PB_LENGTH, 0, crtLength);
  pblock->SetValue(PB_WIDTH, 0, crtWidth);
  pblock->SetValue(PB_HEIGHT, 0, crtHeight);
 	
  pblock->SetValue(PB_GENUVS, 0, TRUE);
	boxObjDesc.MakeAutoParamBlocks(this);	Create IParamBlock2* instance via call to **boxObjDesc.MakeAutoParamBlocks** rather than **CreateParameterBlock**.
		MakeAutoParamBlocks will create IParamBlock2* instances for all ParamBlockDesc2 instances registered on **boxObjDesc** with their P_AUTO_CONSTRUCT flag set, and set it as a reference with the reference index being specified by the ParamBlockDesc2. The IParamBlock2* instance will have default values as specified in the ParamBlockDesc2.

  if (!loading && !GetPhysicalScaleUVsDisabled())
    SetUsePhysicalScaleUVs(true);
}
 	
const int kChunkPolyFix = 0x0100;
	
	  if (!loading && !GetPhysicalScaleUVsDisabled())
	    SetUsePhysicalScaleUVs(true);
	}
	
	const int kChunkPolyFix = 0x0100;	

	bool BoxObject::SpecifySaveReferences(ReferenceSaveManager& referenceSaveManager)
	{
	  // if saving to previous version that used pb1 instead of pb2...
	  DWORD saveVersion = GetSavingVersion();
	  if (saveVersion != 0 && saveVersion <= MAX_RELEASE_R19)
	  {
	     ProcessPB2ToPB1SaveToPrevious(this, pblock2, PBLOCK_REF_NO, descVer1, PBLOCK_LENGTH, CURRENT_VERSION);
	  }
	  return GenBoxObject::**SpecifySaveReferences**(referenceSaveManager);
	}
		This method is for supporting Save To Previous, where the previous version is pb1 based.
		The **ProcessPB2ToPB1SaveToPrevious** function creates an IParamBlock instance, the parameter data is copied from the IParamBlock2* to the IParamBlock*, and then the IParamBlock* is registered to be saved as reference PBLOCK_REF_NO.
		In addition, the IParamBlock* is registered to be stored instead of the IParamBlock2*. This is so that other objects that point at the pb2 will point to the new pb1 when the scene file is loaded. This is needed for things like the scripted controllers which for its Target variables hold a reference and subAnim index. Note that for this to work correctly, the subAnim indices need to be the same for corresponding items in the pb1 and pb2.
		3ds max will store the IParamBlock* instead of the IParamBlock2*, and will delete the IParamBlock*.
		Be sure to call **SpecifySaveReferences()** on the base class.

IOResult BoxObject::Save(ISave *isave)
{
  ULONG nb;
  isave->BeginChunk(kChunkPolyFix);
  isave->Write(&mPolyBoxSmoothingGroupFix, sizeof(bool), &nb);
  isave->EndChunk();
  return IO_OK;
}
 	
IOResult BoxObject::Load(ILoad *iload)
{
	IOResult BoxObject::Save(ISave *isave)
	{
	  ULONG nb;
	  isave->BeginChunk(kChunkPolyFix);
	  isave->Write(&mPolyBoxSmoothingGroupFix, sizeof(bool), &nb);
	  isave->EndChunk();
	  return IO_OK;
	}
	
	IOResult BoxObject::Load(ILoad *iload)
	{	

ParamBlockPLCB* plcb = new **ParamBlockPLCB**(versions, NUM_OLDVERSIONS, &curVersion, this, PBLOCK_REF_NO);	**ParamBlock2PLCB*** plcb = new **ParamBlock2PLCB**(versions, NUM_OLDVERSIONS, &box_param_blk, this, PBLOCK_REF_NO);	**ParamBlock2PLCB** instead of **ParamBlockPLCB**
iload->RegisterPostLoadCallback(plcb);	iload->RegisterPostLoadCallback(plcb);	

boxobj_pb1.cpp	boxobj_pb2.cpp	
// For old Boxes with no kChunkPolyFix, the fix defaults to "off".	// For old Boxes with no kChunkPolyFix, the fix defaults to "off".	
mPolyBoxSmoothingGroupFix = false;	mPolyBoxSmoothingGroupFix = false;	
ULONG nb;	ULONG nb;	
IOResult res = IO_OK;	IOResult res = IO_OK;	
while (IO_OK == (res = iload->OpenChunk())) {	while (IO_OK == (res = iload->OpenChunk())) {	
switch (iload->CurChunkID()) {	switch (iload->CurChunkID()) {	
case kChunkPolyFix:	case kChunkPolyFix:	
iload->Read(&mPolyBoxSmoothingGroupFix, sizeof(bool), &nb);	iload->Read(&mPolyBoxSmoothingGroupFix, sizeof(bool), &nb);	
break;	break;	
}	}	
iload->CloseChunk();	iload->CloseChunk();	
if (res != IO_OK) return res;	if (res != IO_OK) return res;	
}	}	
return IO_OK;	return IO_OK;	
}	}	

boxobj_pb1.cpp

```
void BoxObject::BeginEditParams(IObjParam *ip, ULONG flags, Animatable *prev)
{
  SimpleObject::BeginEditParams(ip, flags, prev);

  if (pmapCreate && pmapParam) {

    // Left over from last Box ceated
    pmapCreate->SetParamBlock(this);
    pmapTypeIn->SetParamBlock(this);
    pmapParam->SetParamBlock(pblock);
    BoxParamDlgProc* dlg = static_cast<BoxParamDlgProc*>(pmapParam->GetUserDlgProc());
    if (dlg != NULL) {
      BOOL rws = dlg->GetRWSState();
      SetUsePhysicalScaleUVs(rws);
    }
  }
  else {

    // Gotta make a new one.
    if (flags&BEGIN_EDIT_CREATE) {
      pmapCreate = CreateCPParamMap(
        descCreate, CREATEDESC_LENGH,
        this,
        ip,
        hInstance,
        MAKEINTRESOURCE(IDD_BOXPARAM1),
        GetString(IDS_RB_CREATIONMETHOD),
        0);

      pmapTypeIn = CreateCPParamMap(
        descTypeIn, TYPEINDESC_LENGH,
        this,
        ip,
        hInstance,
        MAKEINTRESOURCE(IDD_BOXPARAM3),
        GetString(IDS_RB_KEYBOARDENTRY),
        APPENDROLL_CLOSED);
    }

    pmapParam = CreateCPParamMap(
      descParam, PARAMDESC_LENGH,
      pblock,
      ip,
      hInstance,
      MAKEINTRESOURCE(IDD_BOXPARAM2),
      GetString(IDS_RB_PARAMETERS),
      0);
  }

  this->ip = ip;
  if (pmapTypeIn) {
    // A callback for the type in.
    pmapTypeIn->SetUserDlgProc(new BoxTypeInDlgProc(this));
  }
  if (pmapParam) {
    // A callback for the type in.
    pmapParam->SetUserDlgProc(new BoxParamDlgProc(this));
  }
}
```

boxobj_pb2.cpp

```
void BoxObject::BeginEditParams(IObjParam *ip, ULONG flags, Animatable *prev)
{
  SimpleObject::BeginEditParams(ip, flags, prev);

  this->ip = ip;

  // If this has been freshly created by type-in, set creation values:
  if (BoxObject::typeinCreate)
  {
    bool createCube = box_crtype_blk.GetInt(box_create_meth) == 1;
    if (createCube)
    {
      float val = box_typein_blk.GetFloat(box_ti_length);
      pblock2->SetValue(box_length, 0, val);
      pblock2->SetValue(box_width, 0, val);
      pblock2->SetValue(box_height, 0, val);
    }
    else
    {
      pblock2->SetValue(box_length, 0, box_typein_blk.GetFloat(box_ti_length));
      pblock2->SetValue(box_width, 0, box_typein_blk.GetFloat(box_ti_width));
      pblock2->SetValue(box_height, 0, box_typein_blk.GetFloat(box_ti_height));
    }
    typeinCreate = false;
  }

  // throw up all the appropriate auto-rollouts
  boxObjDesc.BeginEditParams(ip, this, flags, prev);
  // if in Create Panel, install a callback for the type in.
  if (flags & BEGIN_EDIT_CREATE)
  {
    box_typein_blk.SetUserDlgProc(new BoxTypeInDlgProc(this));
  }
  // install a callback for the params.
  box_param_blk.SetUserDlgProc(new BoxParamDlgProc(this));
}
```

Pretty much a complete replacement.
box_param_blk.GetUserDlgProc() will return null unless creating a box object after already creating a box object. Otherwise the dialog will not be created until `flags&BEGIN_EDIT_CREATE`.
If **BoxObject::typeinCreate** is true, had hit 'Create' in Keyboard Entry rollout after already creating a box object. In this case, use the type-in parameters for the object parameters.
Call **boxObjDesc.BeginEditParams()** to create rollouts for the ParamBlockDesc2s registered with the class desc. Note that depending on the flags value, not all rollouts will be created
Call **SetUserDlgProc()** on each of the class parameter ParamBlockDesc2s. Do not set DlgProvs on ParamBlockDesc2s for which rollouts will not be created.

boxobj_pb1.cpp	boxobj_pb2.cpp	

void BoxObject::EndEditParams(IObjParam *ip, ULONG flags, Animatable *next)
{
  SimpleObject::EndEditParams(ip, flags, next);
  this->ip = NULL;
 	
  if (flags & END_EDIT_REMOVEUI) {
    if (pmapCreate) DestroyCPParamMap(pmapCreate);
    if (pmapTypeIn) DestroyCPParamMap(pmapTypeIn);
    DestroyCPParamMap(pmapParam);
    pmapParam = NULL;
    pmapTypeIn = NULL;
    pmapCreate = NULL;
  }
  else
  {
    pmapTypeIn->SetUserDlgProc(nullptr);
    pmapParam->SetUserDlgProc(nullptr);
    pmapCreate->SetParamBlock(nullptr);
    pmapTypeIn->SetParamBlock(nullptr);
    pmapParam->SetParamBlock(nullptr);
  }
 	
  // Save these values in class variables so the next object created will inherit them.
  pblock->GetValue(PB_LSEGS, ip->GetTime(), dlgLSegs, FOREVER);
  pblock->GetValue(PB_WSEGS, ip->GetTime(), dlgWSegs, FOREVER);
  pblock->GetValue(PB_HSEGS, ip->GetTime(), dlgHSegs, FOREVER);
}
	void BoxObject::EndEditParams(IObjParam *ip, ULONG flags, Animatable *next)
	{
	  SimpleObject::EndEditParams(ip, flags, next);
	  this->ip = NULL;
	  boxObjDesc.EndEditParams(ip, this, flags, next);
	}
		No need to destroy things, boxObjDesc.EndEditParams takes care of it.
		No need to store current values into static members for next create, handled by IParamBlock2 instances

void BoxObject::SetParams(float width, float height, float length, int wsegs, int lsegs,
  int hsegs, BOOL genUV) {
 	
  pblock->SetValue(**PB_WIDTH**, 0, width);
  pblock->SetValue(PB_HEIGHT, 0, height);
  pblock->SetValue(PB_LENGTH, 0, length);
  pblock->SetValue(PB_LSEGS, 0, lsegs);
  pblock->SetValue(PB_WSEGS, 0, wsegs);
  pblock->SetValue(PB_HSEGS, 0, hsegs);
  pblock->SetValue(PB_GENUVS, 0, genUV);
}
 	
…   lots of code where only these type of changes need to be made …
	void BoxObject::SetParams(float width, float height, float length, int wsegs, int lsegs,
	  int hsegs, BOOL genUV) {
	
	  pblock2->SetValue(**box_width**, 0, width);
	  pblock2->SetValue(box_height, 0, height);
	  pblock2->SetValue(box_length, 0, length);
	  pblock2->SetValue(box_lsegs, 0, lsegs);
	  pblock2->SetValue(box_wsegs, 0, wsegs);
	  pblock2->SetValue(box_hsegs, 0, hsegs);
	  pblock2->SetValue(box_mapping, 0, genUV);
	}
		Switch from using **pblock** to **pblock2**.
		Switch to using enum param IDs (i.e., change **PB_WIDTH** to **box_width**)

int BoxObjCreateCallBack::proc(ViewExp *vpt, int msg, int point, int flags, IPoint2 m, Matrix3& mat) {
  if (!vpt || !vpt->IsAlive())
  {
    // why are we here
    DbgAssert(!_T("Invalid viewport!"));
    return FALSE;
  }
	int BoxObjCreateCallBack::proc(ViewExp *vpt, int msg, int point, int flags, IPoint2 m, Matrix3& mat) {
	  if (!vpt || !vpt->IsAlive())
	  {
	    // why are we here
	    DbgAssert(!_T("Invalid viewport!"));
	    return FALSE;
	  }	

	bool createCube = **box_crtype_blk**.GetInt(box_create_meth) == 1;	Get whether to create a cube from **box_crtype_blk**

Point3 d;
  if (msg == MOUSE_FREEMOVE)
  {
    vpt->SnapPreview(m, m, NULL, SNAP_IN_3D);
  }
 	
  else if (msg == MOUSE_POINT || msg == MOUSE_MOVE) {
    switch (point) {
    case 0:
      sp0 = m;
	Point3 d;
	  if (msg == MOUSE_FREEMOVE)
	  {
	    vpt->SnapPreview(m, m, NULL, SNAP_IN_3D);
	  }
	
	  else if (msg == MOUSE_POINT || msg == MOUSE_MOVE) {
	    switch (point) {
	    case 0:
	      sp0 = m;	

ob->pblock->SetValue(PB_WIDTH, 0, 0.0f);
 	
      ob->pblock->SetValue(PB_LENGTH, 0, 0.0f);
      ob->pblock->SetValue(PB_HEIGHT, 0, 0.0f);
		Unnecessary, P_RESET_DEFAULT set for parameter, default is 0

ob->suspendSnap = TRUE;	ob->suspendSnap = TRUE;	

boxobj_pb1.cpp

```
         p0 = vpt->SnapPoint(m, m, NULL, SNAP_IN_3D);
         p1 = p0 + Point3(.01, .01, .01);
         mat.SetTrans(float(.5)*(p0 + p1));
         {
           Point3 xyz = mat.GetTrans();
           xyz.z = p0.z;
           mat.SetTrans(xyz);
         }
         break;
      case 1:
         sp1 = m;
         p1 = vpt->SnapPoint(m, m, NULL, SNAP_IN_3D);
         p1.z = p0.z + (float).01;
         if (ob->createMeth || (flags&MOUSE_CTRL)) {
           mat.SetTrans(p0);
         }
         else {
           mat.SetTrans(float(.5)*(p0 + p1));
           Point3 xyz = mat.GetTrans();
           xyz.z = p0.z;
           mat.SetTrans(xyz);
         }
         d = p1 - p0;

         square = FALSE;
         if (ob->createMeth) {
           // Constrain to cube
           d.x = d.y = d.z = Length(d)*2.0f;
         }
         else
           if (flags&MOUSE_CTRL) {
             // Constrain to square base
             float len;
             if (fabs(d.x) > fabs(d.y)) len = d.x;
             else len = d.y;
             d.x = d.y = 2.0f * len;
             square = TRUE;
           }

         ob->pblock->SetValue(PB_WIDTH, 0, float(fabs(d.x)));
         ob->pblock->SetValue(PB_LENGTH, 0, float(fabs(d.y)));
         ob->pblock->SetValue(PB_HEIGHT, 0, float(fabs(d.z)));
         ob->pmapParam->Invalidate();

         if (msg == MOUSE_POINT && ob->createMeth) {
           ob->suspendSnap = FALSE;
           return (Length(sp1 - sp0) < 3) ? CREATE_ABORT : CREATE_STOP;
         }
         else if (msg == MOUSE_POINT &&
           (Length(sp1 - sp0) < 3 || Length(d) < 0.1f)) {
           return CREATE_ABORT;
         }
         break;
      case 2:
#ifdef _OSNAP
         p1.z = p0.z + vpt->SnapLength(vpt->GetCPDisp(p0, Point3(0, 0, 1), sp1, m, TRUE));
#else
         p1.z = p0.z + vpt->SnapLength(vpt->GetCPDisp(p1, Point3(0, 0, 1), sp1, m));
#endif
         if (!square) {
           mat.SetTrans(float(.5)*(p0 + p1));
           mat.SetTrans(2, p0.z); // set the Z component of translation
         }

         d = p1 - p0;
         if (square) {
           // Constrain to square base
```

boxobj_pb2.cpp

```
         p0 = vpt->SnapPoint(m, m, NULL, SNAP_IN_3D);
         p1 = p0 + Point3(.01, .01, .01);
         mat.SetTrans(float(.5)*(p0 + p1));
         {
           Point3 xyz = mat.GetTrans();
           xyz.z = p0.z;
           mat.SetTrans(xyz);
         }
         break;
      case 1:
         sp1 = m;
         p1 = vpt->SnapPoint(m, m, NULL, SNAP_IN_3D);
         p1.z = p0.z + (float).01;
         if (createCube || (flags&MOUSE_CTRL)) {
           mat.SetTrans(p0);
         }
         else {
           mat.SetTrans(float(.5)*(p0 + p1));
           Point3 xyz = mat.GetTrans();
           xyz.z = p0.z;
           mat.SetTrans(xyz);
         }
         d = p1 - p0;

         square = FALSE;
         if (createCube) {
           // Constrain to cube
           d.x = d.y = d.z = Length(d)*2.0f;
         }
         else
           if (flags&MOUSE_CTRL) {
             // Constrain to square base
             float len;
             if (fabs(d.x) > fabs(d.y)) len = d.x;
             else len = d.y;
             d.x = d.y = 2.0f * len;
             square = TRUE;
           }

         ob->pblock2->SetValue(box_width, 0, float(fabs(d.x)));
         ob->pblock2->SetValue(box_length, 0, float(fabs(d.y)));
         ob->pblock2->SetValue(box_height, 0, float(fabs(d.z)));
         box_param_blk.InvalidateUI();

         if (msg == MOUSE_POINT && createCube) {
           ob->suspendSnap = FALSE;
           return (Length(sp1 - sp0) < 3) ? CREATE_ABORT : CREATE_STOP;
         }
         else if (msg == MOUSE_POINT &&
           (Length(sp1 - sp0) < 3 || Length(d) < 0.1f)) {
           return CREATE_ABORT;
         }
         break;
      case 2:
#ifdef _OSNAP
         p1.z = p0.z + vpt->SnapLength(vpt->GetCPDisp(p0, Point3(0, 0, 1), sp1, m, TRUE));
#else
         p1.z = p0.z + vpt->SnapLength(vpt->GetCPDisp(p1, Point3(0, 0, 1), sp1, m));
#endif
         if (!square) {
           mat.SetTrans(float(.5)*(p0 + p1));
           mat.SetTrans(2, p0.z); // set the Z component of translation
         }

         d = p1 - p0;
         if (square) {
           // Constrain to square base
```

boxobj_pb1.cpp

```
        float len;
        if (fabs(d.x) > fabs(d.y)) len = d.x;
        else len = d.y;
        d.x = d.y = 2.0f * len;
      }

      ob->pblock->SetValue(PB_WIDTH, 0, float(fabs(d.x)));
      ob->pblock->SetValue(PB_LENGTH, 0, float(fabs(d.y)));
      ob->pblock->SetValue(PB_HEIGHT, 0, float((d.z)));
      ob->pmapParam->Invalidate();

      if (msg == MOUSE_POINT) {
        ob->suspendSnap = FALSE;
        return CREATE_STOP;
      }
      break;
    }
  }
  else
    if (msg == MOUSE_ABORT) {
      return CREATE_ABORT;
    }

  return TRUE;
}

static BoxObjCreateCallBack boxCreateCB;

CreateMouseCallBack* BoxObject::GetCreateMouseCallBack() {
  boxCreateCB.SetObj(this);
  return(&boxCreateCB);
}

BOOL BoxObject::OKtoDisplay(TimeValue t)
{
  return TRUE;
}
```

boxobj_pb2.cpp

```
        float len;
        if (fabs(d.x) > fabs(d.y)) len = d.x;
        else len = d.y;
        d.x = d.y = 2.0f * len;
      }

      ob->pblock2->SetValue(box_width, 0, float(fabs(d.x)));
      ob->pblock2->SetValue(box_length, 0, float(fabs(d.y)));
      ob->pblock2->SetValue(box_height, 0, float(d.z));
      box_param_blk.InvalidateUI();

      if (msg == MOUSE_POINT) {
        ob->suspendSnap = FALSE;
        return CREATE_STOP;
      }
      break;
    }
  }
  else
    if (msg == MOUSE_ABORT) {
      return CREATE_ABORT;
    }

  return TRUE;
}

static BoxObjCreateCallBack boxCreateCB;

CreateMouseCallBack* BoxObject::GetCreateMouseCallBack() {
  boxCreateCB.SetObj(this);
  return(&boxCreateCB);
}

BOOL BoxObject::OKtoDisplay(TimeValue t)
{
  return TRUE;
}
```

boxobj_pb1.cpp	boxobj_pb2.cpp	

// From ParamArray
BOOL BoxObject::SetValue(int i, TimeValue t, int v)
{
  switch (i) {
  case PB_CREATEMETHOD: createMeth = v; break;
  }
  return TRUE;
}
 	
BOOL BoxObject::SetValue(int i, TimeValue t, float v)
{
  switch (i) {
  case PB_TI_LENGTH: crtLength = v; break;
  case PB_TI_WIDTH: crtWidth = v; break;
  case PB_TI_HEIGHT: crtHeight = v; break;
  }
  return TRUE;
}
 	
BOOL BoxObject::SetValue(int i, TimeValue t, Point3 &v)
{
  switch (i) {
  case PB_TI_POS: crtPos = v; break;
  }
  return TRUE;
}
 	
BOOL BoxObject::GetValue(int i, TimeValue t, int &v, Interval &ivalid)
{
  switch (i) {
  case PB_CREATEMETHOD: v = createMeth; break;
  }
  return TRUE;
}
 	
BOOL BoxObject::GetValue(int i, TimeValue t, float &v, Interval &ivalid)
{
  switch (i) {
  case PB_TI_LENGTH: v = crtLength; break;
  case PB_TI_WIDTH: v = crtWidth; break;
  case PB_TI_HEIGHT: v = crtHeight; break;
  }
  return TRUE;
}
 	
BOOL BoxObject::GetValue(int i, TimeValue t, Point3 &v, Interval &ivalid)
{
  switch (i) {
  case PB_TI_POS: v = crtPos; break;
  }
  return TRUE;
}
		Unneeded code

void BoxObject::InvalidateUI()
{
  if (pmapParam) pmapParam->Invalidate();
}
	void BoxObject::InvalidateUI()
	{
	  box_param_blk.InvalidateUI(pblock2->LastNotifyParamID());
	}	

boxobj_pb1.cpp	boxobj_pb2.cpp	

ParamDimension *BoxObject::GetParameterDim(int pbIndex)
{
  switch (pbIndex) {
  case PB_LENGTH:return stdWorldDim;
  case PB_WIDTH: return stdWorldDim;
  case PB_HEIGHT:return stdWorldDim;
  case PB_WSEGS: return stdSegmentsDim;
  case PB_LSEGS: return stdSegmentsDim;
  case PB_HSEGS: return stdSegmentsDim;
  default: return defaultDim;
  }
}
 	
TSTR BoxObject::GetParameterName(int pbIndex)
{
  switch (pbIndex) {
  case PB_LENGTH: return GetString(IDS_RB_LENGTH);
  case PB_WIDTH: return GetString(IDS_RB_WIDTH);
  case PB_HEIGHT: return GetString(IDS_RB_HEIGHT);
  case PB_WSEGS: return GetString(IDS_RB_WSEGS);
  case PB_LSEGS: return GetString(IDS_RB_LSEGS);
  case PB_HSEGS: return GetString(IDS_RB_HSEGS);
  default: return _T("");
  }
}
		Unneeded code

…   lots of code where the only change is changing pblock to pblock2, and using enum param ids ….
{
  if (ip == NULL)
    return;
  BoxParamDlgProc* dlg = static_cast<BoxParamDlgProc*>(pmapParam->GetUserDlgProc());
 	
  dlg->UpdateUI();
}
 	
BOOL BoxObject::GetUsePhysicalScaleUVs()
{
  return ::GetUsePhysicalScaleUVs(this);
}
 	
void BoxObject::SetUsePhysicalScaleUVs(BOOL flag)
{
  BOOL curState = GetUsePhysicalScaleUVs();
  if (curState == flag)
    return;
  if (theHold.Holding())
    theHold.Put(new RealWorldScaleRecord<BoxObject>(this, curState));
  ::SetUsePhysicalScaleUVs(this, flag);
  if (pblock != NULL)
    pblock->NotifyDependents(FOREVER, PART_GEOM, REFMSG_CHANGE);
  UpdateUI();
  macroRec->SetProperty(this, _T("realWorldMapSize"), mr_bool, flag);
}
	{
	  if (ip == NULL)
	    return;
	  BoxParamDlgProc* dlg = static_cast<BoxParamDlgProc*>(box_param_blk.GetUserDlgProc());
	
	  dlg->UpdateUI();
	}
	
	BOOL BoxObject::GetUsePhysicalScaleUVs()
	{
	  return ::GetUsePhysicalScaleUVs(this);
	}
	
	void BoxObject::SetUsePhysicalScaleUVs(BOOL flag)
	{
	  BOOL curState = GetUsePhysicalScaleUVs();
	  if (curState == flag)
	    return;
	  if (theHold.Holding())
	    theHold.Put(new RealWorldScaleRecord<BoxObject>(this, curState));
	  ::SetUsePhysicalScaleUVs(this, flag);
	  if (pblock2 != NULL)
	    pblock2->NotifyDependents(FOREVER, PART_GEOM, REFMSG_CHANGE);
	  UpdateUI();
	  macroRec->SetProperty(this, _T("realWorldMapSize"), mr_bool, flag);
	}
		Get DlgProc from **box_param_blk** instead of **pmapParam**