

# 3ds Max 8 SDK: ParamBlock2 Revisioning

Michaelson Britt  
Software Developer

# ParamBlock2 Revisioning

**Support for adding, removing and changing the type of parameters in a ParamBlock2 between plug-in versions.**

- Prior limitations with reference parameters
  - Resolved limitations
    - Adding parameters
    - Removing parameters
    - Changing parameter type
    - Using P\_VERSION, P\_OBSOLETE and accessing post-load info
  - Remaining limitations
- 
- NOTE: Uses of “ParamBlock” always mean “ParamBlock2” in this presentation. Legacy ParamBlock system is not changed in 3ds Max 8

# Prior limitations with reference parameters

# Prior limitations, and reference parameters

**File loading requires that unique ID of each parameter is not changed between versions**

- New parameters IDs must be added to the end of the enum
- Removed parameters must leave unchanged ID in the enum

**File loading required that reference parameters not be removed or declared in different order between versions**

- System did not support remapping of reference numbers on load
- Reference numbering is determined by order of declaration in the ParamBlockDesc2

**Reference parameters include types where the ParamBlock maintains a reference to separate scene object, and animated parameters, because the ParamBlock maintains a reference to an animation controller**

- TYPE\_MTL
- TYPE\_TEXMAP
- TYPE\_INODE
- etc...

# Prior limitations, cont.

## **Parameter type cannot be changed**

- No built-in handling to migrate parameters between types

## **Parameters cannot be made animatable from non-animatable**

- Adding or removing the P\_ANIMATABLE flag is equivalent to adding or removing a reference parameter, and thus was not allowed

## **Myth: Any changes to a ParamBlockDesc2 would cause legacy files to fail on load**

## **Reality: Previous system would allow the following types of changes**

- UI associations via the p\_ui flag could be added, changed or removed
- Defaults, ranges, accessors and validators with the flags p\_default, p\_range, p\_accessor, and p\_validator could be added, changed or removed
- New parameters could be added, but only to the end of the list

# Resolved limitations

# Resolved limitations

## **File loading now supports remapping of reference numbers on load. User must trigger the ParamBlock2 Revisioning system**

- The new ParamBlock2 must use the P\_VERSION flag
- The old ParamBlock2 must either:
  - Not be using P\_VERSION, or
  - Have a non-matching version number
- The version number is an integer, immediately following the flags field and before the ParamBlock reference number field in the ParamBlockDesc2

## **Remapping of reference numbers**

- Declared order of reference parameters may be changed
- System maintains a mapping from old reference numbers to new
- References are mapped to the old numbering until after the load and post-load callbacks are complete
- The new numbering is used afterwards, all P\_OBSOLETE references are dropped, and reference dependants are notified of the updated numbering

# Resolved limitations: Adding and Deleting Parameters

## Adding parameters

- New reference parameters may be added at the end or between other parameters

## Deleting parameters

- Parameters may be deleted only using the P\_OBSOLETE flag
- Parameter is kept in place, but...
  - Value is reset to default
  - Maintains no references or animation controller
  - Not saved to disk
- The parameter can be removed completely, if the plug-in only loads files where the parameter was already marked P\_OBSOLETE
  - Attempting to load files where the parameter exists but was not marked obsolete is unsafe; hence complete removal of a parameter is not supported when you need to load files from multiple versions prior

**Reminder: user must trigger the ParamBlock2 Revisioning system, using P\_VERSION, for any new handling**

# Resolved limitations: Changing parameter type

## Changing parameter type can be implemented as a remove and an add

- Remove the old version of the parameter using P\_OBSOLETE
- Add the new version parameter
- Use a post-load callback to migrate the value

```
void TestClass::PostLoadCallback( ILoad* iload ) {
    IParamBlock2PostLoadInfo* postLoadInfo = (IParamBlock2PostLoadInfo*)
        pblock->GetInterface( IPARAMBLOCK2POSTLOADINFO_ID );

    // Check for specific obsolete version
    if(postLoadInfo!=NULL && postLoadInfo->GetVersion()==OBSOLETE_VERSION)
    {
        // get the original value
        BOOL onOff = pblock->GetInt(pb_onOff_OBSOLETE, 0, 0);
        //set the original value
        pblock->SetValue( pb_onOff, 0, onOff );

        // see if it has a controller
        Control* onOffCtrl=pblock->GetController(pb_onOff_OBSOLETE,0);

        // if a controller exists add it to the new param
        if( onOffCtrl )
            pblock->SetController(pblock->IDtoIndex(pb_onOff),0,onOffCtrl);

        // NOTE: not necessary to set obsolete params to NULL,
        // or unlink controllers/references
    }
}
```

# Resolved limitations: P\_VERSION

## Using P\_VERSION

- Add flag to the ParamBlockDesc2 flags field, follow immediately with version number, before the reference number
- Previously, version was always set equal to VERSION\_3DSMAX
- Previously, the version could not be retrieved from a saved file, but only from the current ParamBlock; hence, useless for migration

- Setting the current version:

```
static ParamBlockDesc2 test_param_blk (  
    test_params, _T("params"), P_VERSION, &TestClassDesc,  
    P_VERSION + P_AUTO_CONSTRUCT + P_AUTO_UI,  
    VERSION_CURRENT,  
    PBLOCK_REF, //reference number  
    IDD_PANEL, IDS_PARAMS, 0, 0, NULL, //rollout  
    ...
```

- Getting the current version:

```
DWORD newVer = pblock->GetVersion();
```

- Getting a previous version: Use `postLoadInfo->GetVersion()` as above.

# Resolved limitations: P\_OBSOLETE

## Using P\_OBSOLETE

- Add to the flags field for the parameter in the ParamBlockDesc2

```
pb_onOff_OBSOLETE, _T("onOff_OBSOLETE"),  
    TYPE_BOOL_TAB, ONOFF_COUNT, P_OBSOLETE, IDS_ONOFF_OBSOLETE,  
    end,
```

- Certain tags may be removed from obsolete params:
  - p\_ui
  - p\_default, p\_range
  - p\_accessor, p\_validator
- Obsolete params are loaded normally, but after post-load timeframe are reset to default value, and drop any references, which may free objects from memory.
- Obsolete params are never saved to disk

# Resolved limitations: Post-load info

## Post-load info

- Class `IParamBlock2PostLoadInfo`
- Created if `ParamBlock2` Revisioning is triggered during load
- May only be accessed during a `PostLoadCallback`, and is deleted afterwards

```
class TestClassPLCB : public PostLoadCallback {
public:
    TestClassPLCB( TestClass* parent )
        {this->parent=parent;}
    void proc(ILoad *iload)
        {parent->PostLoadCallback(iload); delete this;}
protected:
    TestClass* parent;
};
```

- `GetVersion()` indicates the old `ParamBlock` version
- `GetParamLoaded()` indicates the old param ordering
- `GetParamReorder()` indicates the ordering to preserve reference numbers during load

# Remaining limitations

# Remaining limitations

## **Param IDs may not be changed**

- Never add, delete or move items in the ID enum

## **Reference params may not be deleted**

- Unless the param was already P\_OBSOLETE in any file that will be loaded
- Limitation includes adding or removing P\_ANIMATABLE

## **Param types may not be changed**

- Limitation applies whether param is reference or not
- To change types, add new param and mark previous as obsolete, with a post-load param migration

# Summary

- Use P\_VERSION and update the version number whenever the ParamBlockDesc2 is changed
- Never change Param IDs
- Params may be added or marked obsolete
- Use a post-load callback with the new post-load info mechanism to migrate obsolete params to new params
- Params may not be removed, changed in type, or made animated / non-animated
- Parameter declaration order may be changed if the version number is updated

End of Presentation