

Pimp Your World

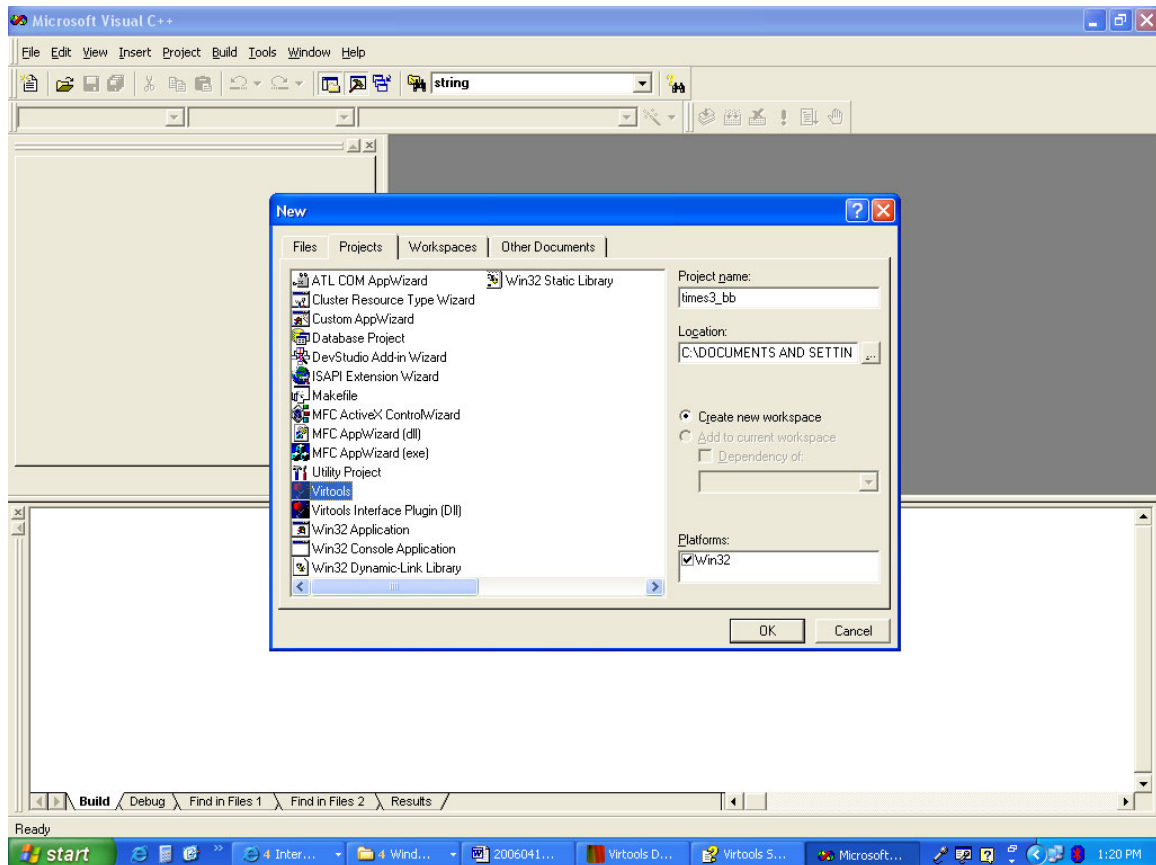
No matter how good [or evil] a given software package might be, intrepid V-ninjas will eventually reach a point where their needs begin to outstrip its capabilities. Good applications provide various means by which users can extend and customize the platform's native functionality. Virtools offers several ways to do this, one of the most important of which is through the Virtools SDK (software development kit).

Let's begin by brewing our very own building block. For the time being, we'll begin with an exceedingly simple one: [drumroll] The Times3_BB! That's right folks, a building block that will multiply ANY integer you might pass it by three!

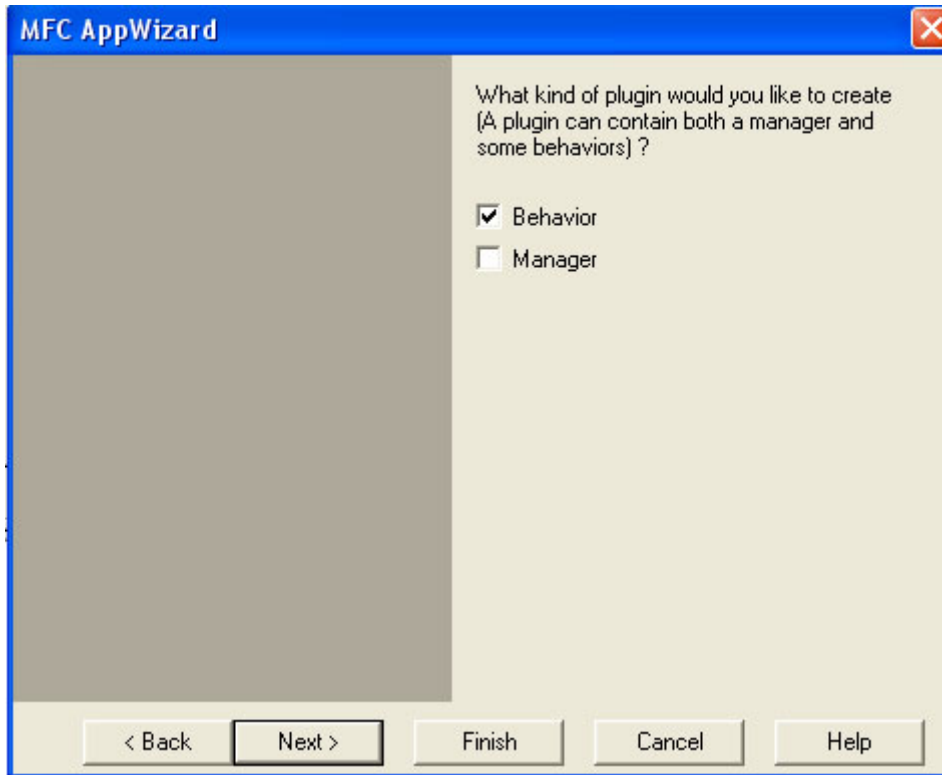
To begin, the dark secret of developing VT BB's is that they provide a wizard that writes almost all of the code for you. This App wizard is named "Virtools Behaviors AppWizard.awx" and is found in the "Utils" directory of the Virtools SDK and should be placed, along with the other predefined App Wizards, in the directory:

C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Template

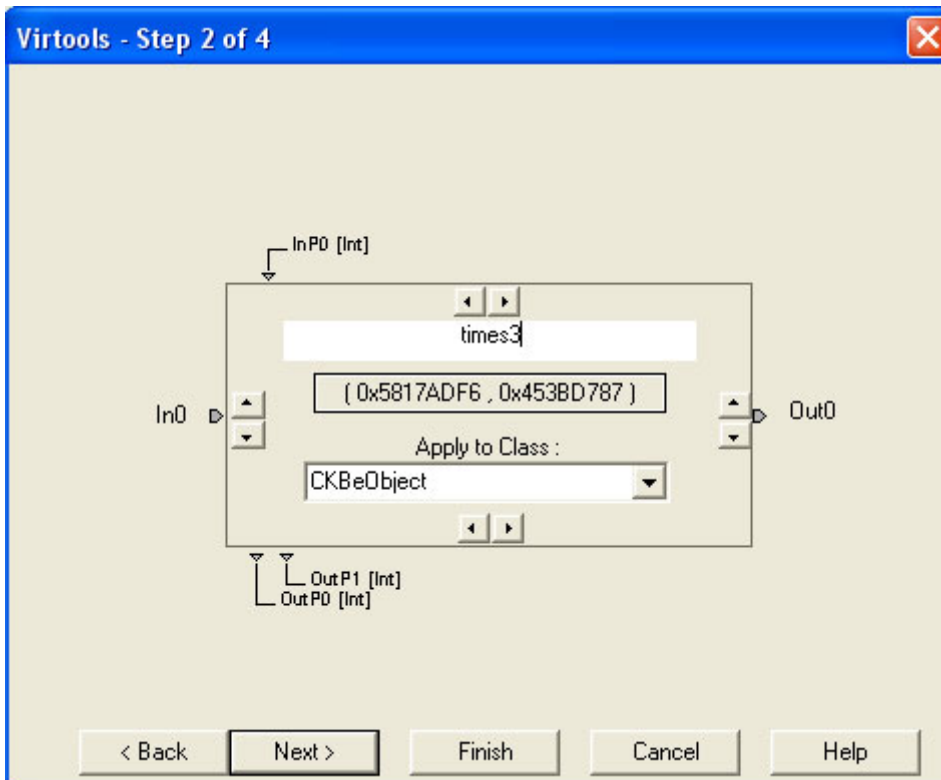
Once you do this, when you open up a new workspace in Visual Studio 6.0, you will see your beloved Virtools icon. Select this, name your project, and make sure it's going somewhere sensible to start the wizard.



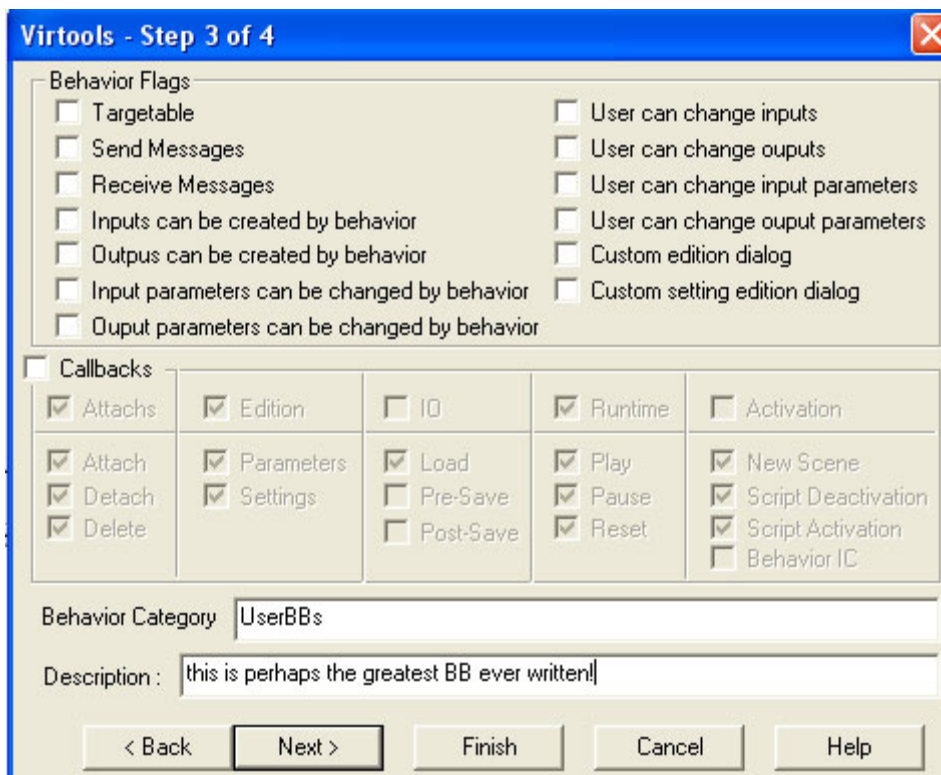
For this next step, you almost always want just a behavior. Managers perform system-wide operations such as collision management.



Next is a very important step wherein you define what your BB is going to look like. Here you give it a name, and set up Bin's, Bout's, Pin's and Pout's, including their respective data types.

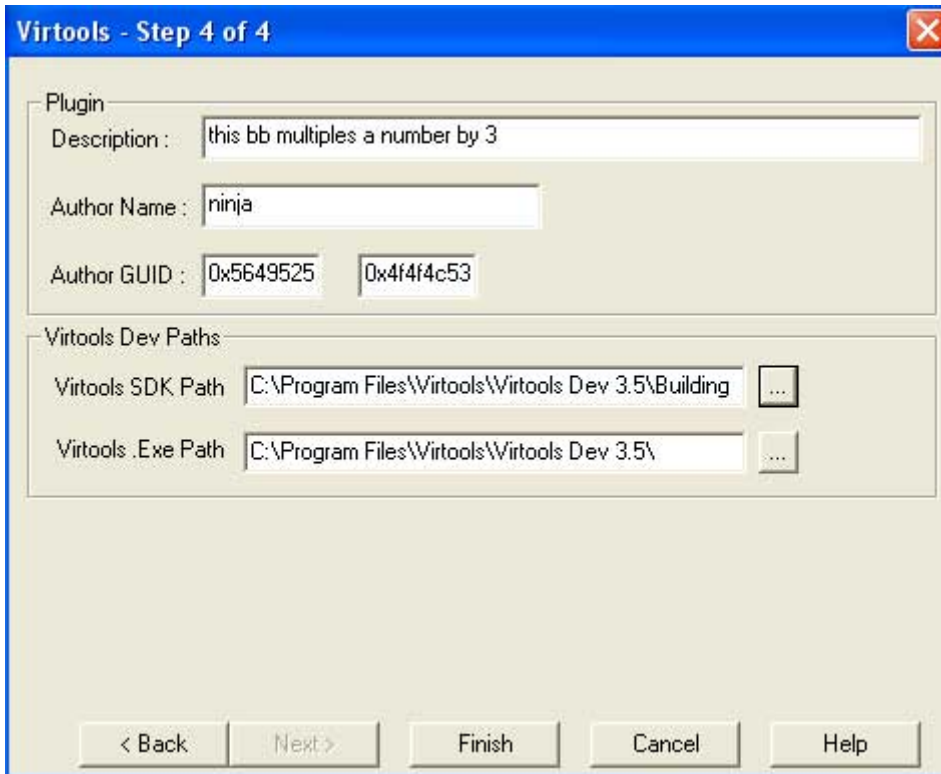


In this case I just added a single Pout by clicking on the right arrow at center bottom.



Now you set various flags that will control how much code Virtools generates for you. Note: we're going to keep it simple for the time being, but note, the Callbacks section will become very important as you start doing more sophisticated BB's.

Check the paths so that your bb dll gets put in the right place [the Building Blocks directory].



When you click finish you'll see that Virtools generates a project for you. Let's look at some of this code.

The first code file you'll see is call [bb_name]_bb.cpp. In our case, it looks like this:

```
// times3_bb.cpp : Defines the initialization routines for the plugin DLL.
//
#include "CKAll.h"

#define PLUGIN_COUNT 1 // A Behavior

CKPluginInfo g_PluginInfo[PLUGIN_COUNT];

int CKGetPluginInfoCount() {
    return PLUGIN_COUNT;
}

CKPluginInfo* CKGetPluginInfo(int Index)
```

```

{
    int Plugin = 0;
    g_PluginInfo[Plugin].m_Author          = "ninja";
    g_PluginInfo[Plugin].m_Description    = "this bb multiples a number by
3";
    g_PluginInfo[Plugin].m_Extension      = "";
    g_PluginInfo[Plugin].m_Type          =
CKPLUGIN_BEHAVIOR_DLL;
    g_PluginInfo[Plugin].m_Version        = 0x00010000;
    g_PluginInfo[Plugin].m_InitInstanceFct = NULL;
    g_PluginInfo[Plugin].m_GUID          =
CKGUID(0x5817ADF6,0x453BD787);
    g_PluginInfo[Plugin].m_Summary        = "Enter your summary";
    Plugin++;

    return &g_PluginInfo[Index];
}

```

```

// Additional behaviors definitions should be added here....
// CKObjectDeclaration* MyBehaviorDecl();

```

```

// This function should be present and exported for Virtools
// to be able to retrieve objects declarations.
// Virtools will call this function at initialization
void RegisterBehaviorDeclarations(XObjectDeclarationArray *reg)
{
    RegisterBehavior(reg, FillBehaviortimes3Decl);
// Additional behaviors registrations should be added here....
// RegisterBehavior(reg, MyBehaviorDecl);
}

```

```

// If no manager is used in the plugin
// these functions are optional and can be exported.
// Virtools will call 'InitInstance' when loading the behavior library
// and 'ExitInstance' when unloading it.
// It is a good place to perform Attributes Types declaration,
// registering new enums or new parameter types.

```

```

/*

```

```

CKERROR InitInstance(CKContext* context)
{
    return CK_OK;
}

```

```

CKERROR ExitInstance(CKContext* context)
{

```

```

    return CK_OK;
}

*/

```

The code in this section of your project is basically just getting called when Virtools loads, and it give the environment information about the BB for inclusion in its list of Building Block. It is important to note however, the last two functions `InitInstance` and `ExitInstance`, which can be good places to put initialization and cleanup code respectively.

The next and most important file to investigate is usually called `[bb_name].cpp`. This is where most of your actual function code will go. Specifically in the function:

```
int times3(const CKBehaviorContext& BehContext)
```

times3.cpp

```

////////////////////////////////////
//                               times3
////////////////////////////////////
#include "CKAll.h"

CKObjectDeclaration *FillBehaviortimes3Decl();
CKERROR Createtimes3Proto(CKBehaviorPrototype **);
int times3(const CKBehaviorContext& BehContext);

CKObjectDeclaration *FillBehaviortimes3Decl()
{
    CKObjectDeclaration *od = CreateCKObjectDeclaration("times3");

    od->SetType(CKDLL_BEHAVIORPROTOTYPE);
    od->SetVersion(0x00010000);
    od->SetCreationFunction(Createtimes3Proto);
    od->SetDescription("this is perhaps the greatest BB ever written!");
    od->SetCategory("UserBBs");
    od->SetGuid(CKGUID(0x5817ADF6,0x453BD787));
    od->SetAuthorGuid(CKGUID(0x56495254,0x4f4f4c53));
    od->SetAuthorName("ninja");
    od->SetCompatibleClassId(CKCID_BEOBJECT);

    return od;
}

CKERROR Createtimes3Proto(CKBehaviorPrototype** pproto)
{

```

```

        CKBehaviorPrototype *proto = CreateCKBehaviorPrototype("times3");
        if(!proto)      return CKERR_OUTOFMEMORY;

//---  Inputs declaration
        proto->DeclareInput("In0");

//---  Outputs declaration
        proto->DeclareOutput("Out0");

//----- Input Parameters Declaration
        proto->DeclareInParameter("InP0",CKPGUID_INT);

//---  Output Parameters Declaration
        proto->DeclareOutParameter("OutP0",CKPGUID_INT);
        proto->DeclareOutParameter("OutP1",CKPGUID_INT);

//----  Local Parameters Declaration

//----  Settings Declaration

        proto->SetFlags(CK_BEHAVIORPROTOTYPE_NORMAL);
        proto->SetFunction(times3);

        *ppproto = proto;
        return CK_OK;
}

int times3(const CKBehaviorContext& BehContext)
{
    return CKBR_OK;
}

```

So to add our desired functionality all we need to do is add the following code above "return CKBR_OK" in the function above.

//This code give us a pointer to the current behavioral context, which allows you to refer to all the objects etc in your scene. We then deactivate the Bin.

```

CKBehavior* beh = BehContext.Behavior;
beh->ActivateInput(0,FALSE);

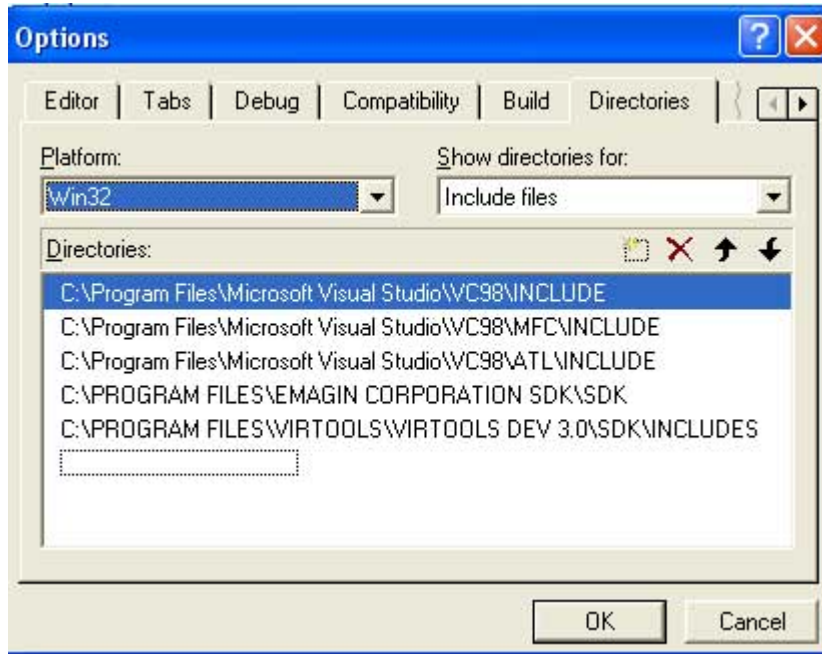
//Now we want to grab the value of our pIn
int input;
beh->GetInputParameterValue(0,&input);

//now let's multiply it by 3!
Input=input*3;

//And send it out, and activate the bOut
beh->SetOutputParameterValue(0, &input);
beh->ActivateOutput(0);

```

Finally, before we try to compile, be sure to add the right include and lib directories to your project.



Then “rebuild all” and compile. At this point you should be able to open up Virtools and see your brand new BB in the “user BB’s” category of your behavior manager. Try it out, and prepare to be astounded!