

Ken's Visual C++ / trueSpace eXtension Helper Site

How to set up Microsoft Visual C++ 6 for use with the trueSpace 5.1 SDK

1. [Download the trueSpace SDK from Caligari](#) and unzip it. Make a note of where you put the SDK files. You'll need to know that later.
2. You now need to tell Visual C++ where the trueSpace SDK files are. Start Visual C++ and select **Options** from the **Tools** menu when it is finished loading.
3. Click on the **Directories** tab and select **Include Files** from the **Show directories for** pop-up menu.
4. Click the **New** button. It's the transparent box with the star.
5. An empty entry should have appeared in the list with a button next to it. This is the **Browse** button. Go ahead and click it..
6. Navigate to where you unzipped the SDK files, select the **headerfiles** subdirectory, and then click **OK** to close the dialog.
7. Now select **Library Files** from the **Show directories for** pop-up menu.
8. Click the **New** button and then the **Browse** button that appears next to the new entry.
9. Navigate to the SDK directory again, but this time select the **libraries** subdirectory. Click **OK** to close the dialog.
10. Finally click **OK** to close the **Options** dialog.

That is all there is to setting up Visual C++. To test that you successfully completed the steps above, open the workspace for blanktsx sample distributed with the trueSpace SDK and compile it. If you receive no errors or warnings, you have successfully completed this tutorial.

How to create and set up a trueSpace TSX project in Visual C++ 6

If you haven't done so already, please complete [VC Tutorial #1](#) before starting this tutorial.

This tutorial will show how to create a basic TSX from scratch. [Part 1](#) shows how to create an empty Visual C++6 project and adjust the necessary project settings. [Part 2](#) shows how to add the necessary resources to the tsxframe project and [Part 3](#) explains the five functions necessary to every TSX. At the end of [Part 3](#) you should have a completed tsxframe project that can be used as a starting point for your own TSX projects. If you don't wish to complete Parts 1-3, you can download the final tsxframe project's files [here](#). [Part 4](#) of this tutorial shows you how to modify the tsx project into a final TSX.

Part 1: Creating an empty DLL project and adjusting its settings

1. If you have closed Visual C++, go ahead and reopen it.
2. Select **New** from **File** menu.
3. Make sure that the **Projects** tab is active and select **MFC AppWizard (dll)** from the available project types. I know that some of you reading this probably don't like using the AppWizard. I find that for DLLs it doesn't kludge up the code too much and makes adding dialogs later a little easier. Regardless, if you know how to write DLLs without AppWizard, you should be able to figure out from this tutorial how to set up a TSX project from scratch.
4. Since we are just going to create a basic TSX framework with this tutorial, enter **tsxframe** for the **Project name**. Enter the directory where you would like to store the

- project and make sure that **Create new workspace** is active. Click **OK** to continue to the next step.
5. On Step 2 of the AppWizard, leave the settings at the default and click **Finish** to continue.
 6. On the **New Projects Information** dialog, just click **OK** to continue.
 7. Now there are some adjustments that need to be made to the tsxframe project settings. Select **Settings** from the **Project** menu.
 8. Click on the **Link** tab.
 9. Change the extension on the **Output file name** to **tsx**.
 10. Add **tsxapi.lib** to the **Object/library modules** field.
 11. Make sure you have applied these changes to both the Debug and Release versions of the tsxframe project. When you are done, click **OK** to close the **Project Settings** dialog.
 12. One final thing needs to be done to complete this part of the tutorial. An **#include** needs to be added to the project for the **tsx.h** file. Open up the **testframe.h** file from either **FileView** or the **File** menu. Under the line **#include "resource.h"** add the line **#include "tsx.h"**
 13. Make sure that you save your work at this point. Do a **Save All** and a **Save Workspace** from the **File** menu.

You now have an empty dll project with all of the necessary settings adjusted to enable it to compile with the trueSpace SDK. [Part 2](#) of this tutorial shows how to add the resources needed for every TSX. [Part 3](#) explains the five trueSpace API functions necessary to every TSX. [Part 4](#) shows how to modify the tsxframe project into a final TSX.

Part 2: Adding resources to the TSX project

This part of the tutorial will build upon the TSX project that was started in [Part 1](#). If you haven't completed it yet, please do so now.

This part of the tutorial will show how to add the resources necessary to every TSX. These include a button graphic and a help string.

1. The first resource that needs to be added to the project is the bmp file that the TSX will use for it's button graphic inside of trueSpace. For this tutorial the bitmap will be imported into the project instead of created inside of VC++. This graphic should be 30x22 pixels and any color depth. Once you have a suitable graphic, name it **tsxbutton.bmp** and copy it into the tsxframe's project directory.
2. If you have completed Part 1 of this tutorial, now is the time to open the tsxframe project's workspace in Visual C++.
3. Once the project is loaded, go to **ResourceView**, right-click the folder labeled **tsxframe resources** and click **Import...** from the pop-up menu.
4. Now navigate to the **tsxbutton.bmp** file, select it and then click **Import** to continue.
5. There should now be bitmap resource for the button graphic. Right-click on this resource and select **Properties** from the pop-up menu.
6. Change the **ID** field to **IDB_TSXBUTTON** and then close the dialog.
7. The second resource that needs to be added is a string table. This string table is going to contain the help string that trueSpace will display when a user hovers over the TSX's button. To insert the string table, right-click the folder labeled **tsxframe resources** in **ResourceView** and select **Insert...** from the pop-up menu.

8. Choose **String Table** from the list of available resource types. Click **New** to create the string table and close the dialog.
9. Double-click on the **String Table** to open it up in the String Table Editor.
10. Now double-click the first line in the string table. A **String Properties** dialog should have opened. For **ID** enter **IDS_TSXHELP** and for caption enter **TSX help place holder**.
11. Save your work at this point. Do a **Save All** and a **Save Workspace** from the **File** menu.

The bitmap and the string table are the only two resources we will be adding to the tsxframe project in this tutorial. In a future tutorial, other resources like menus and dialogs will be added. The next part of this tutorial, [Part 3](#), explains the five trueSpace API functions necessary to every TSX. [Part 4](#) shows how to modify the tsxframe project into a final TSX.

Part 3: The five required trueSpace API functions

This part of the tutorial will build upon the tsxframe project that was started in [Part 1](#) and [Part 2](#). If you haven't completed them yet, please do so now.

As you can tell from this section's title, there are five trueSpace API functions that every TSX is required to implement and export. These functions are `tsxGetData`, `tsxOnLeftClick`, `tsxOnRightClick`, `tsxDeactivate`, and `tsxGetVersion`. Code listings for each of these functions are given below. While I give brief descriptions of each of these functions, more information can be found in the trueSpace SDK manual. The comments in the code below are pulled directly from the samples that come with the trueSpace SDK.

You can either copy and paste the code into the testframe project or manually type it in. In either case, all of the following code should be entered into the **testframe.cpp** file below the line **CTsxframeApp theApp**; Each function's code should be copied directly below the one described before it.

1. The first necessary function is **tsxGetData**. This function is the first function called by trueSpace. trueSpace uses this function to register your TSX and get its name, the name of its developer, the ID of its button graphic, and the ID for its help string. trueSpace returns a value through this function which is stored in **g_tsxid**. This ID uniquely identifies the TSX to trueSpace and prevents the TSX from loading on more than one button. The ID is also used with callback functions. Here is the code listing:

```
// Saved tsxid for one installation of this extension
// Note: Using single global this will prevent it from being installed on more
// than one button.
int g_tsxid =0;

// First function called from trueSpace, on loading the tSX
int tsxGetData (tsxData* tsx_data, int tsxid)
{
// Put at beginning of exported functions if using MFC,
// or your plugin is likely to crash
AFX_MANAGE_STATE(AfxGetStaticModuleState());
```

```

static tsxDATA MyTsxData =
{ "tsx framework", "Plugin Developer", IDB_TSXBUTTON, IDS_TSXHELP};

*tsx_data = MyTsxData;

// Save tsxid for this extension, for use with Callback functions.
g_tsxid = tsxid;

return tsxPLUG_DONE;
}

```

2. The second function your tsx needs to implement is the **tsxOnLeftClick** function. trueSpace calls this function when the TSX's button is clicked with the left mouse button.

```

// this function will be called when the tSX button is left-clicked
int tsxOnLeftClick()
{
// Put at beginning of exported functions if using MFC,
// or your plugin is likely to crash
AFX_MANAGE_STATE(AfxGetStaticModuleState());

// nothing functional here now

return tsxPLUG_DONE;
}

```

3. The third function is **tsxOnRightClick**. This function is similar to the **tsxOnLeftClick** function from above except it is called when the TSX's button is clicked with the right button. A good use for this function is implementing settings or about dialog windows.

```

// this function will be called when the tSx button is right-clicked
void tsxOnRightClick()
{
// Put at beginning of exported functions if using MFC,
// or your plugin is likely to crash
AFX_MANAGE_STATE(AfxGetStaticModuleState());

// nothing functional here now
}

```

4. The fourth function is the **tsxDeactivate** function. This function is called by trueSpace when a TSX is deactivated. This can occur when the TSX's button is left-clicked again, some other tool is selected, or the user exits trueSpace. This function is a good place to close windows or perform any memory cleanups necessary.

```

// This function will be called whenever the tSx is deactivated
// - User left-clicks on the tSX button
// - User selects some other tool

```

```
// - User exits trueSpace
void tsxDeactivate()
{
// Put at beginning of exported functions if using MFC,
// or your plugin is likely to crash
AFX_MANAGE_STATE(AfxGetStaticModuleState());

// nothing functional here now
}
```

5. The final function that a TSX needs is **tsxGetVersion**. This function queries trueSpace for the version of the trueSpace API. If the version number returned is lower than that specified by the TSX's author, the user will receive an error stating that the TSX was developed for another version of trueSpace.

```
// new for tS5, this function allows plugin authors to specify which
// version of the API they need:
// 500 is version 5.0, 430 is version 4.3, etc.
int tsxGetVersion(int tsxid)
{
// Put at beginning of exported functions if using MFC,
// or your plugin is likely to crash
AFX_MANAGE_STATE(AfxGetStaticModuleState());

return 510;
}
```

6. Now that these functions have been added to the project, they need to be exported. To do this, the **tsxframe.def** file needs to be modified. Go ahead and open it now from **FileView**. Under **Exports** add these lines:

```
tsxGetData
tsxOnLeftClick
tsxOnRightClick
tsxDeactivate
tsxGetVersion
```

While **tsxframe.def** is still open, go ahead and change the **Library** value to **"tsxframe"** and the **Description** value to **"tsxframe trueSpace tsx"**.

7. Save your work at this point. Do a **Save All** and a **Save Workspace** from the **File** menu.

Believe it or not, the tsxframe project is now a fully functional TSX. You should be able to compile it and load it into trueSpace without any errors or warnings. Of course it doesn't do much yet. The next part of this tutorial, [Part 4](#), shows how to add some basic functionality to the tsxframe and change its settings so that it can truly be called a TSX.

Part 4: Changing the tsxframe project into a final TSX

This is the final part of this VC Tutorial #2. Here you will modify the [final tsxframe project](#)

[files](#) into what can truly be called a TSX, if only a simple one. If you have not completed [Part 1](#), [Part 2](#), and [Part 3](#), please complete them now, or download the [final tsxframe project files](#).

This last part of the tutorial will focus on the changes that need to be made to the tsxframe project to create a final TSX. I want to use this part primarily to show you which project settings need to be changed for the final TSX. We will be adding some basic functionality, but only for the purposes of saying we completed a final TSX.

1. Some file manipulation needs to be done before we start messing with the tsxframe project. If the tsxframe project is open, go ahead and close it now.
2. Using Windows Explorer, navigate to the directory where the tsxframe project folder is. Copy the entire tsxframe directory and rename the copy **addcube**.
3. Now a unique button graphic needs to be added to the project. I've already created one that you can [download here](#). It's not a work of art, but it'll do for now. Place it in the **addcube** directory. If Windows asks, overwrite the old tsxbutton.bmp.
4. Now either double-click the **tsxframe.dsw** in the **addcube** directory to open it. Alternatively, you could open it from the Visual C++ file menu. Make sure that you open the **tsxframe.dsw** file in the **addcube** directory and not the tsxframe directory.
5. Select **Settings** from the **Project** menu and click the **Link** tab when the dialog appears.
6. Change the **Output file name:** field to **addcube.tsx** for both the Release and Debug configurations and click **OK** to close the Settings dialog.
7. Go to **FileView** and open the **tsxframe.def** file. Change **tsxframe** to **addcube** in both the **Library** and **Description** strings.
8. Now open the **String Table** from **ResourceView** and double-click **IDS_TSXHELP** to open the **String Properties** dialog. Change the **Caption** field to "Add a unit cube at the origin". Close the dialog.
9. Now open the **VS_VERSION_INFO** file. You are not going to modify anything in this file, but I wanted to point it out. This is where Windows gets the property information when you right click a program. While any of these key's values can be changed, keys can only be added or deleted in the lower group. If you are going to distribute your TSX it is a good idea to edit this file with the correct information.
10. Finally, some functional code needs to be added to the addcube TSX. It wouldn't be a very good TSX if it didn't actually do something. You are going to add this code directly into the **tsxOnLeftClick** function. As you recall, this function is in the **testframe.cpp** file. Open it now from **FileView**. Paste the following code over the line **"// nothing functional here now"** in the **tsxOnLeftClick** function.

```
tsxPOLYHEDRON* pPolyh = tsxCreateCube(1, 1.0f, 1.0f,1.0f);
tsxGNodeSetName((tsxGNODE*) pPolyh, "Unit Cube");
tsxSceneAddObject((tsxGNODE*) pPolyh, e_tsxFALSE);
```

Don't worry too much about how this code works. Essentially, it creates an empty cube with a resolution of 1 and x,y,z dimensions of 1. It then renames the cube and adds it to the current scene.

11. Save your work at this point. Do a **Save All** and a **Save Workspace** from the **File** menu.
12. Now build the addcube project. Copy the **addcube.tsx** (from either the Debug or Release folder depending on the project settings) into the trueSpace **tsx** directory.

13. Launch trueSpace and load the addcube TSX. When you hover over the addcube button, you will notice that the help string we changed is displayed in the status bar. When you click the addcube button, a small cube should have been added to the origin of the current scene.

Congratulations for making all the way through this tutorial! I know it was kind of dry, but hopefully you now have all of the knowledge you need to use Visual C++ 6 to write trueSpace eXtensions. The knowledge to use the trueSpace API effectively is a whole different beast. I hope to post some tutorials on that sometime in the future.