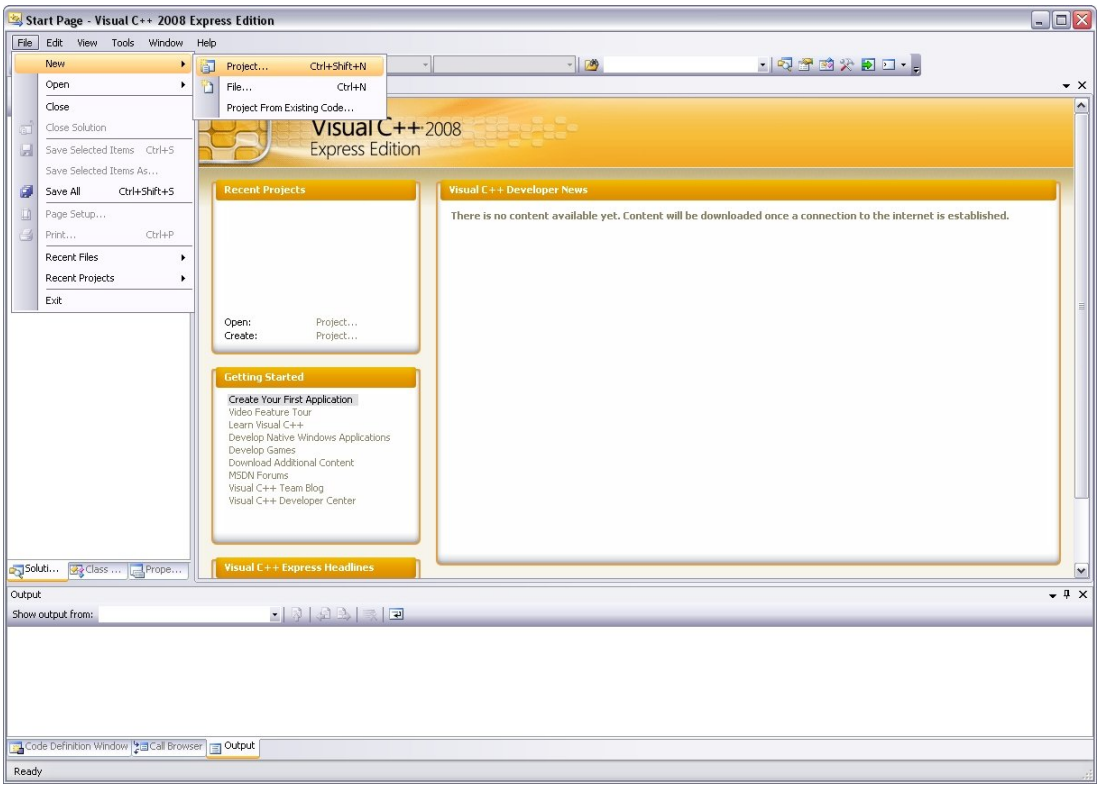
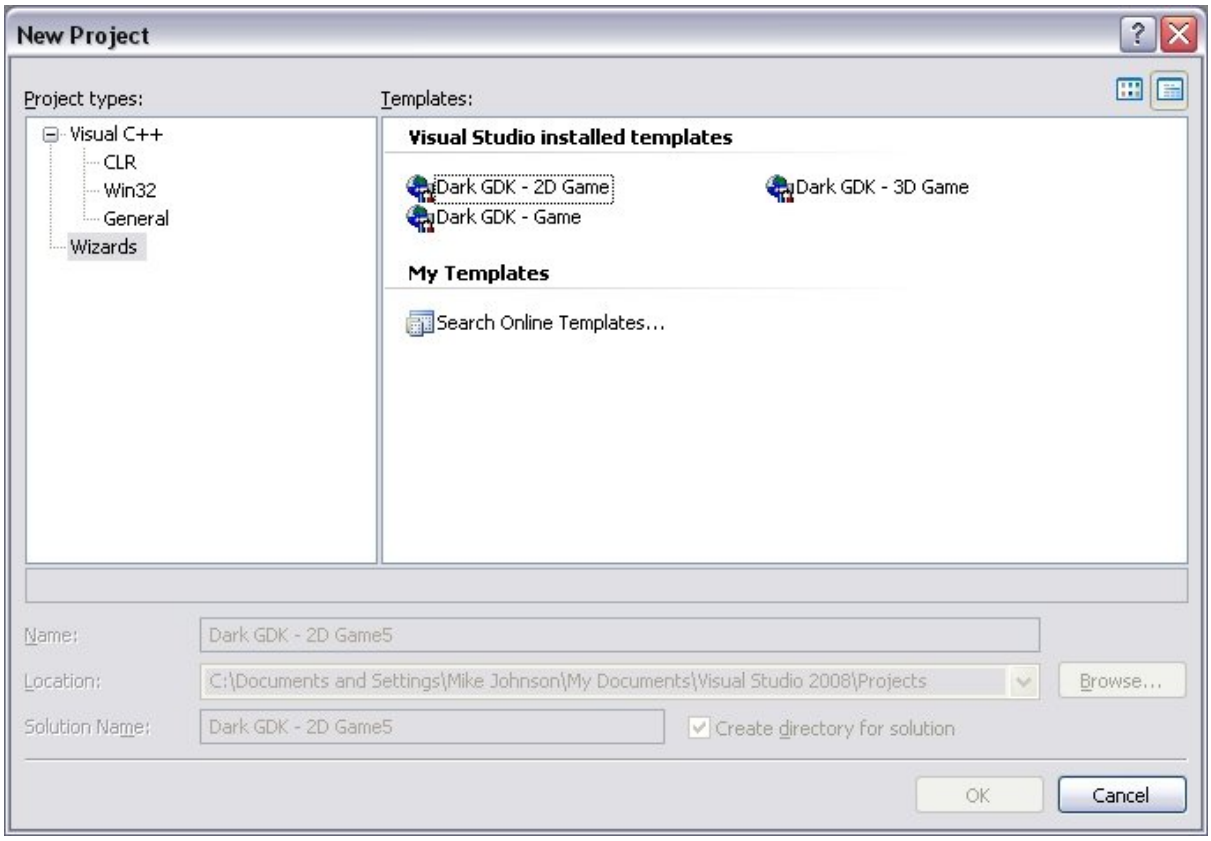
Your first program using DarkGDK

Introduction
In this tutorial we're going to guide you through creating your first program using Dark GDK. The aim of this program is to display an animated image on screen.

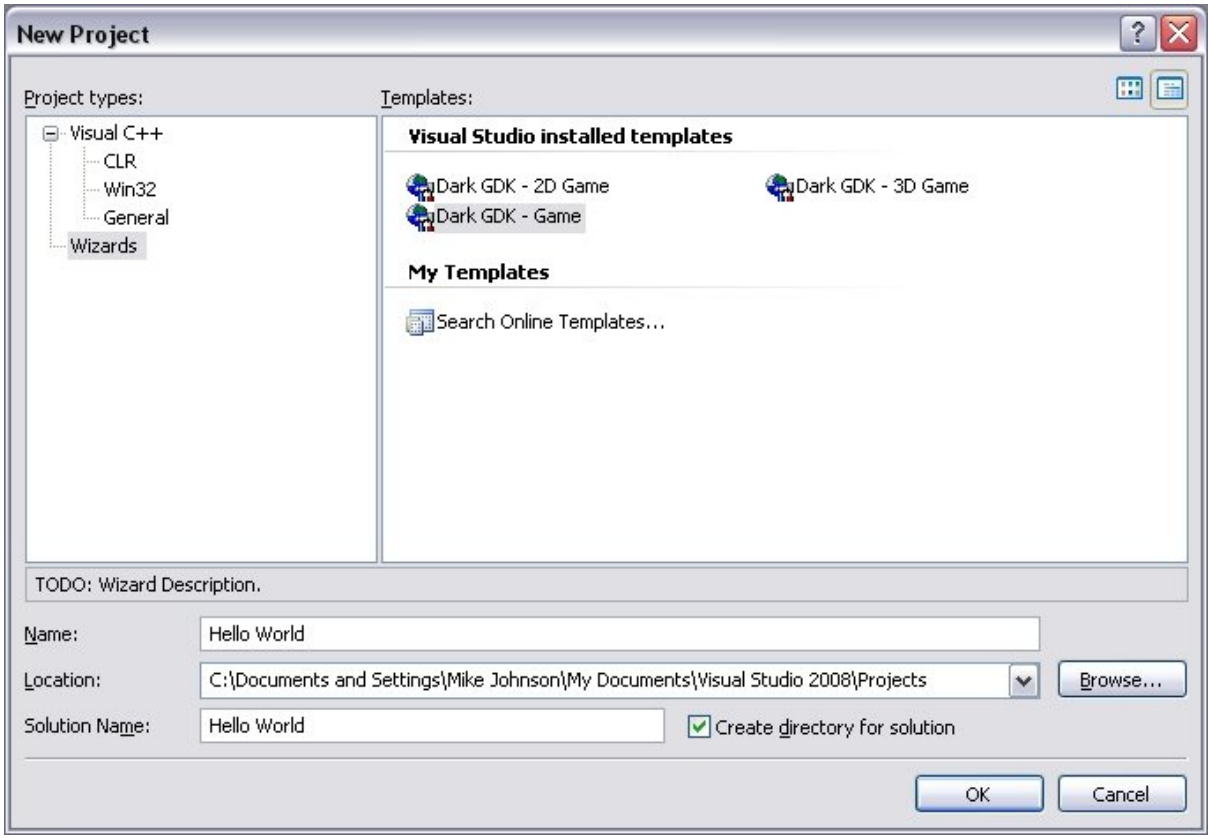
Creating a project
Launch Microsoft Visual C++ 2008 and go to the File menu and select New and then Project.



You will now be presented with options for creating a new project.

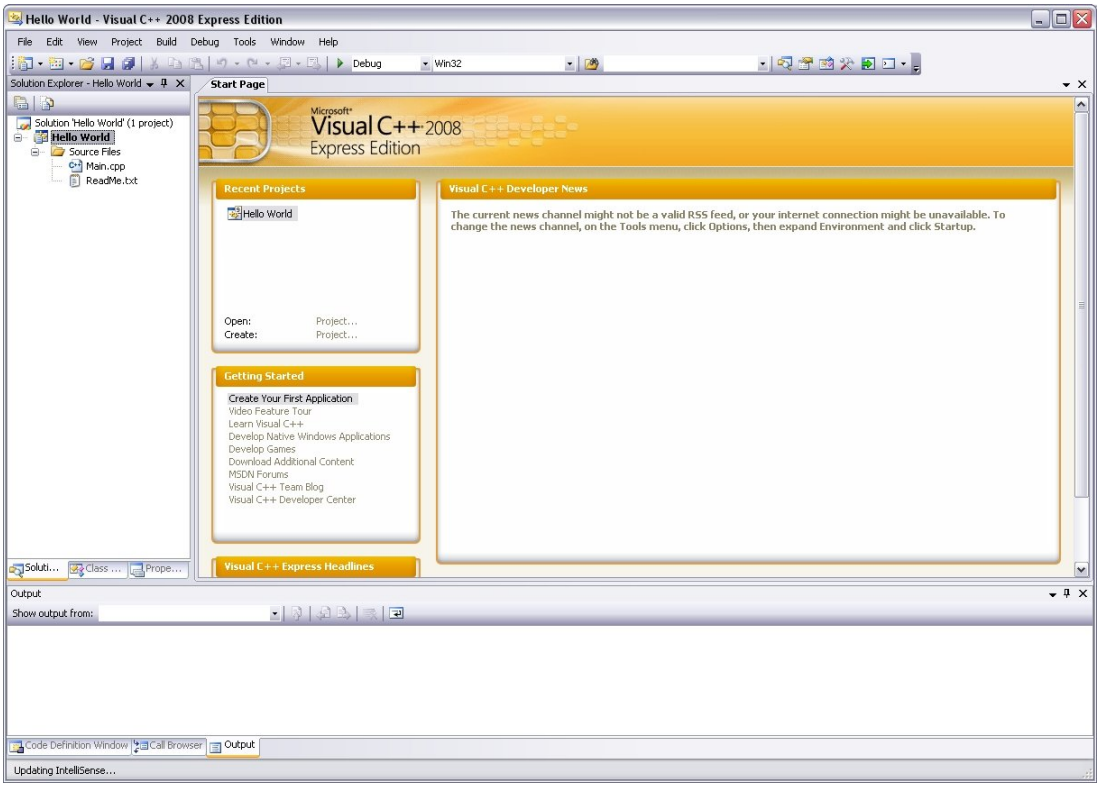


On the left side of the dialog is a list of available project types. Make sure you select Wizards. Now look over to the right side and you will see a list of installed templates. This list will include several templates for Dark GDK. Select *Dark GDK – Game* and enter *Hello World* for the project name and leave the location at the default directory.

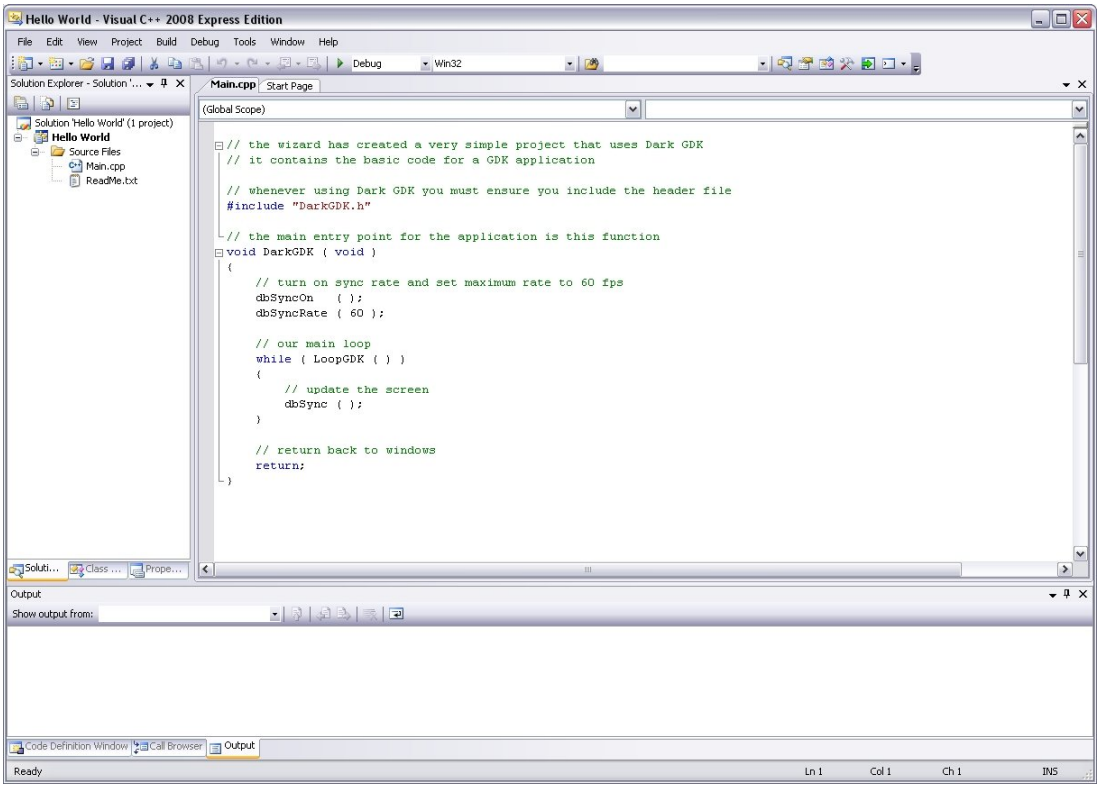


Now click on the OK button and the project will be created.

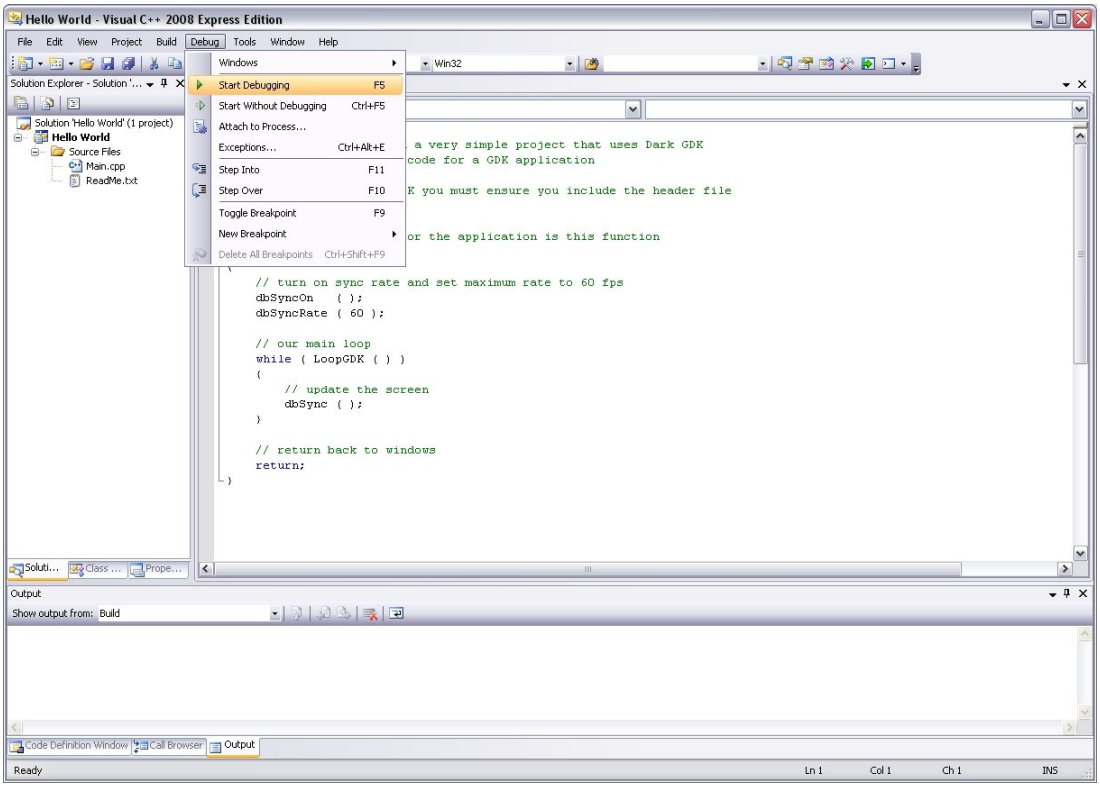
Working with the project
The Dark GDK – Game template will create a project and add a source file named Main.cpp. Double click on this file in the Solution Explorer.



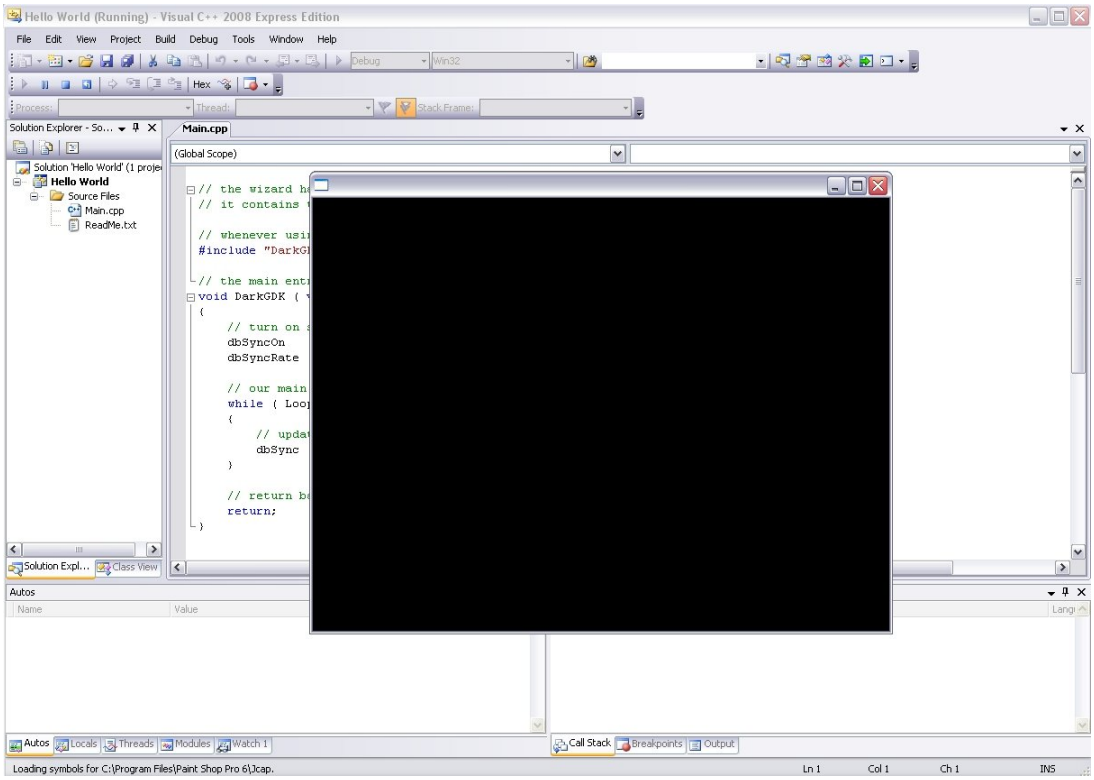
The main window now contains the contents of Main.cpp. It contains the most basic code required for a Dark GDK program.



Try running the project by going to the Debug menu and selecting Start Debugging



When the debug process starts the code will be compiled and the program will run. At this stage it isn't going to do a great deal. Press escape to quit out.



Initial source code

When we created this project some source code has been generated and placed into Main.cpp. It is important to understand this initial code. Let's examine each part of it:

```
#include "Dark GDK.h"
```

This line is used to include the Dark GDK header file into your source file. Any program that relies on Dark GDK must include this header file.

```
void Dark GDK ( void )
```

A program that uses Dark GDK does not have a WinMain or main function. Instead the entry point for the program is this function. The program will start and end in this function.

```
dbSyncOn ( );
```

This line is used to control the way the screen updates. Basically by calling this function we're saying we want to control when the screen is updated. Later on in our main loop we'll see how we can update the screen.

```
dbSyncRate ( 60 );
```

Another option is available after setting sync on. Using dbSyncRate we can control the maximum frame rate of our program. In this instance a value of 60 is used meaning the program will run at a maximum of 60 frames per second. We could use a value of 30 to indicate a maximum of 30 frames per second or a special value of 0 to allow the program to run as fast as possible.

```
while ( LoopGDK ( ) )
```

This while loop has been set up to act as our main program loop. The reason for calling LoopGDK is to allow Dark GDK to perform some internal functionality while the program is running. LoopGDK will return a value of 1 while the program can run. If the user was to click on the close button of the window then LoopGDK will return a value of 0 in which case the while loop would exit. It is recommended to set up your main loop in this way.

```
dbSync ( );
```

The final line of code is inside the while loop and it is a call to the function dbSync. This function will update the contents of the screen. As an example, say you have some sprites set up – they will not appear on screen until you call this function.

Adding functionality

Now that you understand the basic code it's time to start adding our own functionality to make our program. As mentioned earlier the aim of this program is to have an image on screen that animates.

Dark GDK contains many options for displaying graphics on screen. For our example we can use the sprite functions. The sprite functions contained in Dark GDK allow you to place images on screen, rotate, scale, move them around, animate and more. They are an ideal option for working with 2D graphics. We're going to use some of the sprite functions for our program.

First of all we're going to need an image to display on screen. We can use one that comes with the Dark GDK. Open Windows Explorer and navigate to the directory where you installed Dark GDK. By default this will be C:\Program Files\The Game Creators\Dark GDK. Once here go into the Tutorials folder and now into the Media folder and now open animatedsprite.png.

This image has been set up in such a way that it contains frames of animation. By displaying part of the image at one time and then another part and so on we can play back an animation. The image also contains an alpha channel. This means that when it is drawn to screen certain parts of it will not get drawn. In our image the black areas will not be drawn to screen.

Let's copy this image into our project directory. Copy the file and paste it into your project directory e.g. C:\Documents and Settings\Mike\My Documents\Visual Studio 2008\Projects\Hello World\Hello World. With the image in place we can load it into our program.

Go back into your source code and add a new line before our main loop. In this line we're going to load the image file. The method of loading image files is quite simple. We can call the function dbLoadImage. This function takes two parameters. The first is the filename of the image. In our case this is "animatedsprite.png". The second parameter is an ID number. ID numbers are used in Dark GDK as a way of keeping track of resources. The vast majority of functions contained in Dark GDK use ID numbers. Each function set has its own set of ID numbers e.g. image function ID numbers are separate from sprite function ID numbers. As an example say we used an ID number of 1 when loading an image. Later on we want to delete the image. We can do this by calling dbDeleteImage and passing in our ID number. For our example we'll use an ID number of 1:

```
dbLoadImage ( "animatedsprite.png", 1 );
```

After calling this function our image will be in memory but now we need a way of drawing it to the screen. We can draw it to the screen using the sprite functions. To

create a sprite we can call the function dbSprite. This function takes four parameters. The first parameter is an ID number, the second parameter is a position on the X axis, the third parameter is a position on the Y axis and finally the fourth parameter is an ID number for the image we want to attach to the sprite.

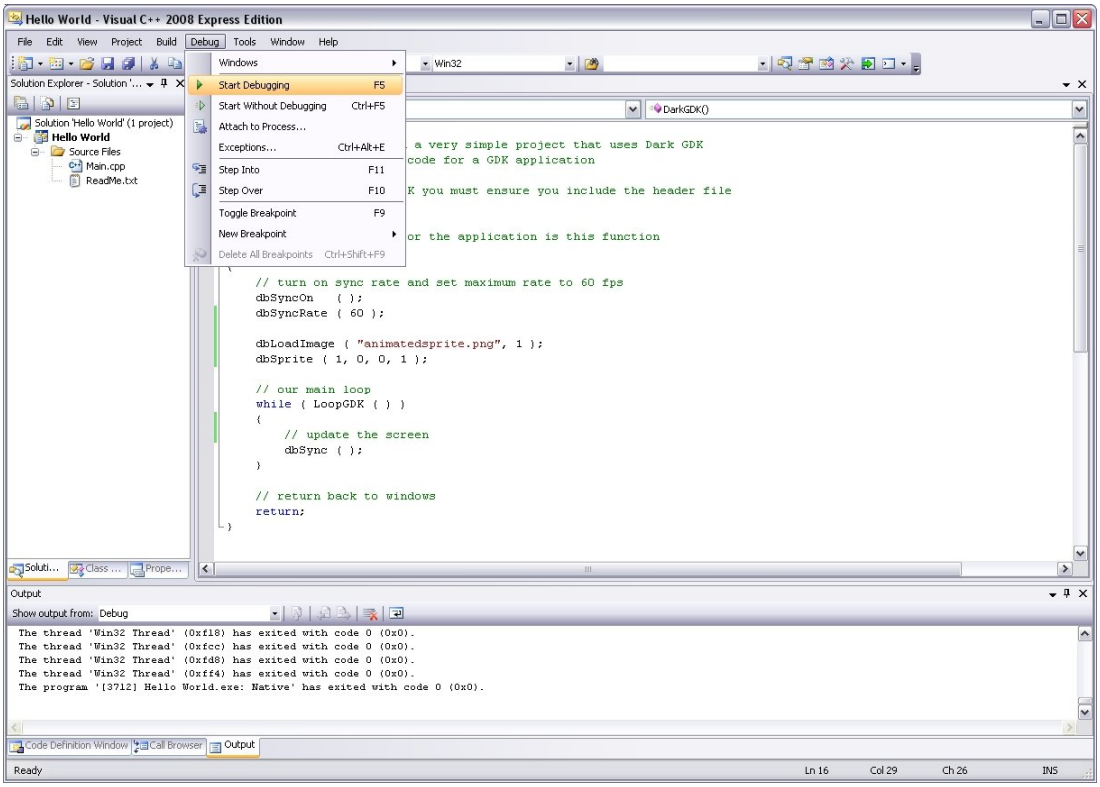
Bare in mind that ID numbers for sprites are separate to those of images. Therefore we can use an ID number of 1 for our sprite. The next two parameters define the location on screen. For now we'll place our sprite at the top left of the screen which is 0, 0. We have an image loaded into ID 1 so we can pass in this value for the image ID. Our call to dbSprite looks like this:

```
	dbSprite ( 1, 0, 0, 1 );
```

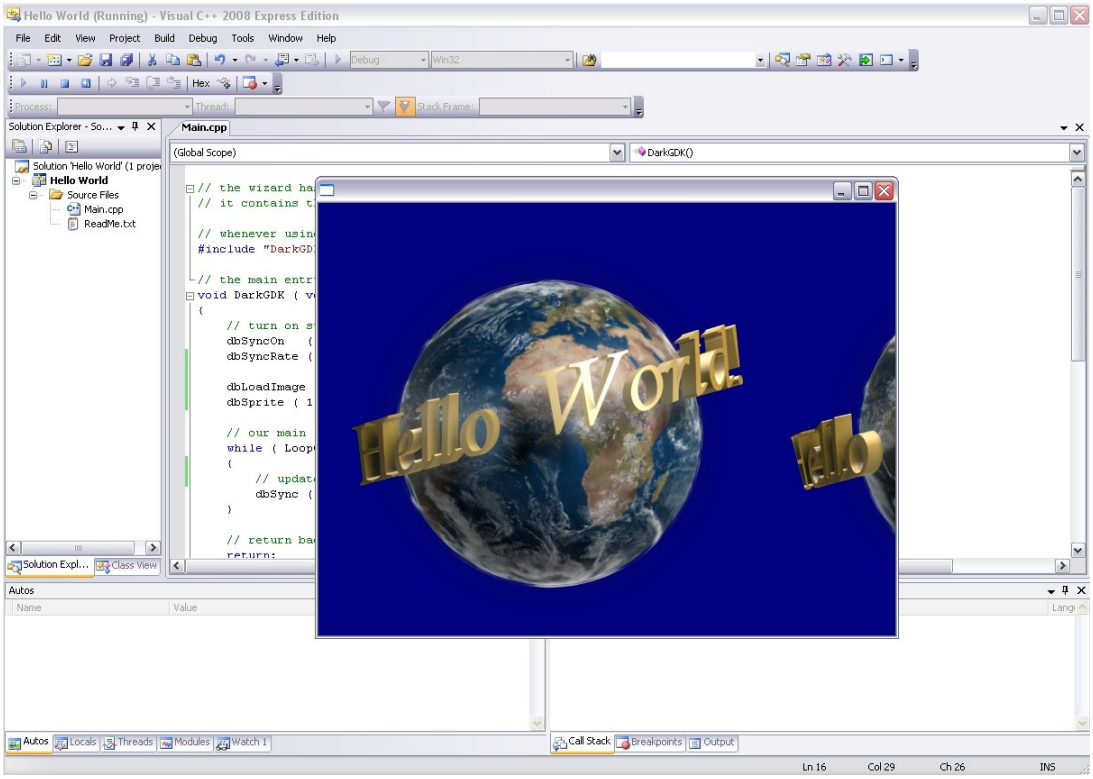
Before trying our changes lets take a look at the modified code:

```
#include "DarkGDK.h"

void DarkGDK ( void )
{
	dbSyncOn   ( );
	dbSyncRate ( 60 );

	dbLoadImage ( "animatedsprite.png", 1 );
	dbSprite ( 1, 0, 0, 1 );

	while ( LoopGDK ( ) )
	{
		dbSync ( );
	}

	return;
}
```

Now go to the Debug menu and select Start Debugging.



When the program runs you will see the following.



We now have our image on screen but it isn't quite right. The whole image is being displayed but our window isn't big enough to display it so a large part of it is cut off. What we instead want to do is only show a portion of the image at one time and then

move to another portion and so on. By doing this we'll create an animated sprite. Making these changes is quite simple.

Modify your code and remove the two new lines we put in. We're going to replace those lines with a call to another function. This function is named dbCreateAnimatedSprite. By using this function you can take the image and create a sprite that will cycle through each part of the image. The function takes several parameters:

- sprite ID, this is the ID number of the sprite
- filename, the filename of the image to load
- across, the number of frames going across
- down, the number of frames going down
- image ID, this is the ID number of the image

For the sprite ID we can use a value of 1. We know the filename will be "animatedsprite.png". Now lets refer back to our image before setting the values for across and down

Take a look at how the image is set out. It is one large image but it contains multiple smaller images. Each of these smaller images can be referred to as a frame. Going across the image we can see 4 frames and going down we also have 4. This results in 16 frames. We can display each frame in turn for our animation. Now we know that for the across and down parameters we can use a value of 4.

The final parameter is the ID number for the image. We can use a value of 1 for this.

Our call to dbCreateAnimatedSprite now looks like this:

```
dbCreateAnimatedSprite ( 1, "animatedsprite.png", 4, 4, 1 );
```

This function call will set the sprite up and get everything in place. However, at this point we haven't defined properties for the sprite such as its position. Therefore it will

not be drawn on screen. To solve this we simply add a call to dbSprite below it to define our position:

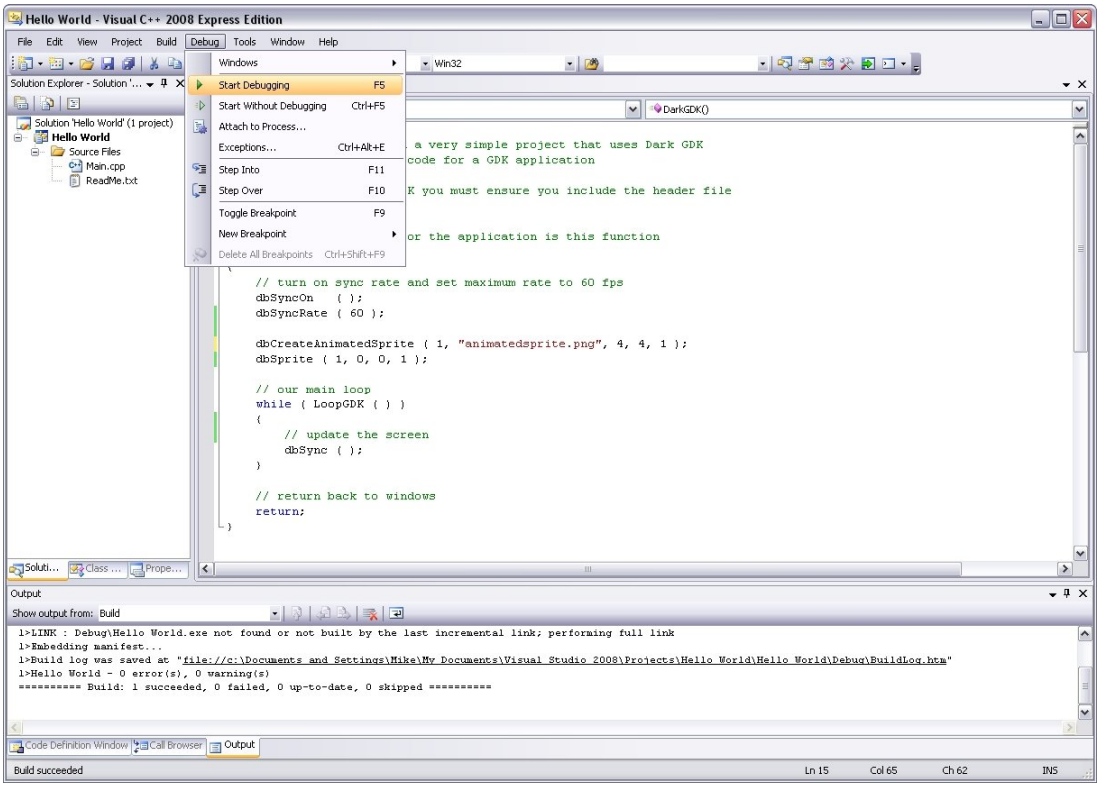
```
dbSprite ( 1, 0, 0, 1 );
```

We created the sprite with an ID number of 1 so for this call we use the same ID, The second and third parameters are the X and Y coordinates. Just like before we'll place the sprite at 0, 0 meaning the top left of the screen. Finally we need an image ID. When calling dbCreateAnimatedSprite we used an image ID of 1 so for this we also use 1.

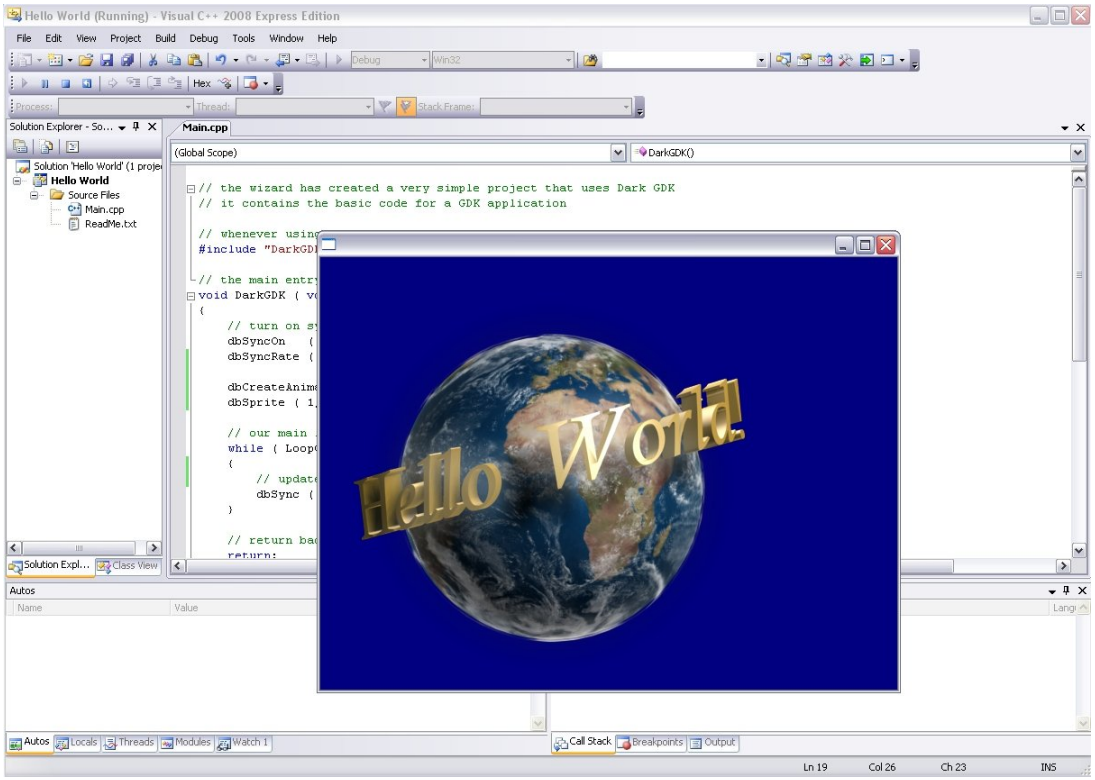
Lets now review our changes:

```
#include "DarkGDK.h"

void DarkGDK ( void )
{
	dbSyncOn   ( );
	dbSyncRate ( 60 );

	dbCreateAnimatedSprite ( 1, "animatedsprite.png", 4, 4, 1 );
	dbSprite ( 1, 0, 0, 1 );

	while ( LoopGDK ( ) )
	{
		dbSync ( );
	}

	return;
}
```

Now go to the Debug menu and select Start Debugging:



When the program runs you will see the following:



The main change here is that only part of the image is being shown. This is exactly what we want as it means we can now cycle through the frames in the image. The final step is to adjust which part of the image is shown on the sprite. We can achieve this by calling dbPlaySprite. This function takes four parameters. The first

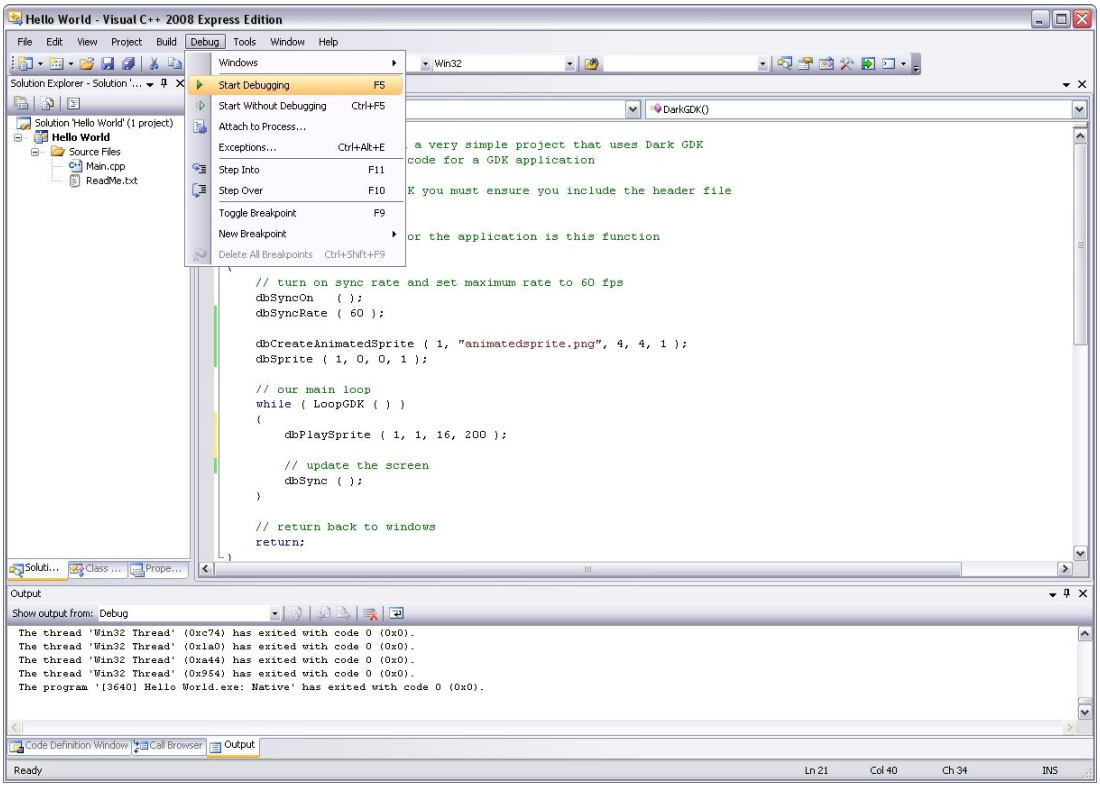
parameter is the ID number of the sprite, the second parameter is the start frame, the third parameter is the end frame with the last parameter being the time delay between switching to the next frame. We can call dbPlaySprite as follows:

```
dbPlaySprite ( 1, 1, 16, 200 );
```

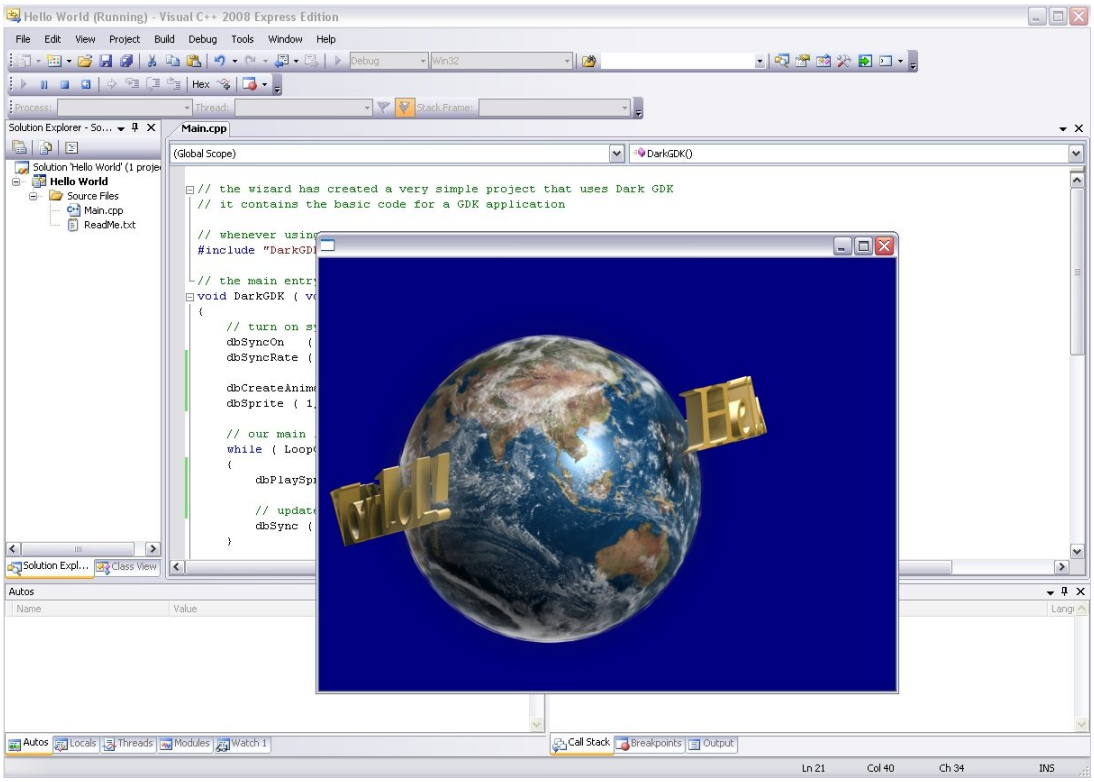
We use a value of 1 for the initial parameter as this is the ID of the sprite we want to control. A value of 1 is used for the start frame. This will result in the animation starting from the 1st frame in the image. We have 4 frames across and 4 frames down in the image resulting in a total of 16 images. If we want to play the animation to the last frame then we pass in a value of 16 for the end frame. For the final parameter a value of 200 is used. This will result in a delay of 200 ms before the sprite displays the next frame. This new line needs to be added inside the main loop:

```
#include "DarkGDK.h"

void DarkGDK ( void )
{
	dbSyncOn   ( );
	dbSyncRate ( 60 );

	dbCreateAnimatedSprite ( 1, "animatedsprite.png", 4, 4, 1 );
	dbSprite ( 1, 0, 0, 1 );

	while ( LoopGDK ( ) )
	{
		dbPlaySprite ( 1, 1, 16, 200 );
		dbSync ( );
	}

	return;
}
```

Now everything is ready to run. Go to the Debug menu and select Start Debugging:



This time when the program runs the sprite will switch between the frames of animation:



Conclusion

With a few simple steps we've been able to create a program that displays an animated image on screen.