# Controlling CDGS/CDTS X7 Applications From Other Processes

CorelDRAW Graphics Suite (CDGS) X7 and CorelDRAW Technical Suite (CDTS) X7 applications provide a number of ways for extending their functionality with custom 3rd-party plugins and macros.

Solutions based on VBA Macros, VSTA plugins and C++ plugins all run in the context of the host applications such as CorelDRAW, Corel PHOTO-PAINT, Corel DESIGNER. However sometimes it is necessary to talk to CDGS/CDTS applications from another stand-alone application. This tutorial explores the ways of connecting to CDGS/CDTS applications from outside process, which could be a native C++ application or a managed .NET program written in C# or Visual Basic.

The exact way of communicating with CDGS/CDTS applications depends on the software framework and programming language used to create the 3rd-party application. All of the examples below create a simple command-line (console) application which connect to CorelDRAW X7 and create a simple artistic text object “Hello, world” in 12 pt, Arial font. These examples are equivalent to the following VBA macro:

**Private** **Sub** CreateTextInCorelDRAW(text **As** **String**, fontName **As** **String**, \_

fontSize **As** **Single**)

  ActiveLayer.CreateArtisticText 0, 0, text, Font:=fontName, Size:=fontSize

**End** **Sub**

**Sub** Main()

  CreateTextInCorelDRAW "Hello, world!", "Arial", 24

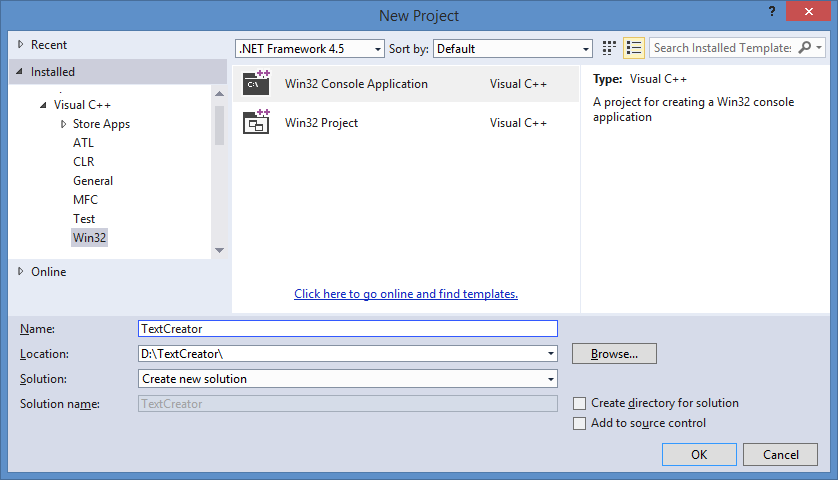
**End** **Sub**

## Native C++

### Native C++ with low-level COM

This is the most low-level interface between two applications where higher-level frameworks are not available or not desirable. You can write a native C++ application and use pure COM APIs to connect to an instance of CDGS/CDTS application.

To start, create a simple C++ Win32 Console Application in Visual Studio by selecting File > New > Project… and picking “Win32 Console Application” from “Visual C++ > Win32” project template category:



Enter the project name and its location and click OK. Create the project with default settings (or tweak them as needed).

Once the solution is created, open stdafx.h file and add #include <windows.h> at the bottom to include some of the Windows platform API definitions that will be required later in this tutorial.

The first thing that the application needs in order to talk to CorelDRAW is the description of its objects and interfaces. This information is provided by CorelDRAW’s type library which is installed on the system when CDGS/CDTS is installed. Visual C++ provides a special directive to import the type information from the application type library, #import.

So, add the following command at the top of main .cpp file:

#import "libid:95E23C91-BC5A-49F3-8CD1-1FC515597048" version("11.5") \

      raw\_interfaces\_only no\_smart\_pointers \

      rename("GetCommandLine", "VGGetCommandLine") \

      rename("CopyFile", "VGCopyFile") \

      rename("FindWindow", "VGFindWindow")

Here we import a type library with the ID 95E23C91-BC5A-49F3-8CD1-1FC515597048 which is VGCore type library which describes the types provided by CorelDRAW and Corel DESIGNER. The version, “11.5” specifies the version of type library to import, which becomes important if you have multiple versions of CDGS and/or CDTS installed on the machine. The actual version string corresponds to the application version but in hexadecimal form. 11 hexadecimal equals to 17 decimal, so version “11.5” refers to type library version 17.5.

By default, Visual Studio generates a number of helper classes to make using COM objects easier, but since the goal of this article is to use low-level COM interfaces, we add “raw\_interfaces\_only” and “no\_smart\_pointers” options for #import directive. Finally, CorelDRAW/Corel DESIGNER object model has methods such as GetCommandLine, CopyFile and FindWindow which conflicts with macros defined by Win32 API, so here we rename them to VBGetCommandLine and so on to eliminate the name conflict.

When compiling the solution, all the COM interfaces from VGCore type library are imported and converted into a regular C++ header file, VGCoreAuto.tlh.

Now that all the type information from CorelDRAW/Corel DESIGNER type library is imported and is available to your C++ program, implement the main() function of your console application:

**int** wmain(**int** argc, **wchar\_t**\* argv[])

{

  HRESULT hr = CoInitializeEx(**nullptr**, COINIT\_APARTMENTTHREADED);

**if** (**SUCCEEDED**(hr))

  {

    hr = CreateTextInCorelDRAW(L"Hello, world", L"Arial", 24.0f);

**if** (**FAILED**(hr))

    {

      wprintf(L"Error occurred: 0x%08X\n", hr);

    }

    CoUninitialize();

  }

**return** 0;

}

Since the native application uses COM for communication with CorelDRAW/Corel DESIGNER, Windows COM sub-system must be initialized first by using CoInitializeEx() Windows API function and then uninitialized at the end with CoUninitialize().

CreateTextInCorelDRAW() is the main function that connects to CorelDRAW X7. It utilizes CoCreateInstance() Windows API to create an instance of the main Application object of CorelDRAW. This function requires a class ID (CLSID) which can be obtained from “CorelDRAW.Application.17” ProgID using CLSIDFromProgID() API.

It is important to specify the actual class ID (directly using CLSID, or by converting a ProgID string) because VGCore type library is shared between all vector graphics applications in CDGS/CDTS (CorelDRAW, CorelDRAW Essentials, Corel DESIGNER) and they can be installed on the same computer at the same time. So your application must be able to choose whether it wants to work, say, with CorelDRAW or Corel DESIGNER.

HRESULT CreateTextInCorelDRAW(**const** **wchar\_t**\* text, **const** **wchar\_t**\* fontName,

**float** fontSize)

{

  CLSID clsid;

  HRESULT hr = CLSIDFromProgID(L"CorelDRAW.Application.17", &clsid);

**if** (**FAILED**(hr))

**return** hr;

**VGCore**::IVGApplication\* app = **nullptr**;

  hr = CoCreateInstance(clsid, **nullptr**, CLSCTX\_LOCAL\_SERVER,

**\_\_uuidof**(**VGCore**::IVGApplication),

**reinterpret\_cast**<**void**\*\*>(&app));

**if** (**FAILED**(hr))

**return** hr;

  hr = CreateText(app, text, fontName, fontSize);

  app->Release();

**return** hr;

}

This function creates an instance of CorelDRAW Application class and obtains IVGApplication COM interface to it which can be used for communication with CorelDRAW.

If CorelDRAW is already running, CoCreateInstance() will connect to the running instance of CorelDRAW. If not, CorelDRAW will be launched automatically before the function returns, however, in this case, CorelDRAW will be started hidden and no UI will appear by default (which could be useful to perform some background graphics processing without showing the application main Window to the user).

Finally, CreateText() helper function creates the artistic text object in the document of CorelDRAW, using the app object just created.

HRESULT CreateText(**VGCore**::IVGApplication\* app, **const** **wchar\_t**\* text,

**const** **wchar\_t**\* fontName, **float** fontSize)

{

  HRESULT hr = app->put\_Visible(**VARIANT\_TRUE**);

**if** (**FAILED**(hr))

**return** hr;

**VGCore**::IVGDocument\* doc = **nullptr**;

  hr = app->get\_ActiveDocument(&doc);

**if** (**FAILED**(hr))

**return** hr;

**if** (doc == **nullptr**)

  {

    hr = app->CreateDocument(&doc);

**if** (**FAILED**(hr))

**return** hr;

  }

**VGCore**::IVGLayer\* active\_layer = **nullptr**;

  hr = doc->get\_ActiveLayer(&active\_layer);

**if** (**SUCCEEDED**(hr))

  {

    BSTR bstr\_text = SysAllocString(text);

    BSTR bstr\_font = SysAllocString(fontName);

**VGCore**::IVGShape\* shape = **nullptr**;

    hr = active\_layer->CreateArtisticText(

      0.0, 0.0, bstr\_text, **VGCore**::cdrTextLanguage::cdrLanguageMixed,

**VGCore**::cdrTextCharSet::cdrCharSetMixed, bstr\_font, fontSize,

**VGCore**::cdrTriState::cdrUndefined, **VGCore**::cdrTriState::cdrUndefined,

**VGCore**::cdrFontLine::cdrMixedFontLine,

**VGCore**::cdrAlignment::cdrLeftAlignment, &shape);

    SysFreeString(bstr\_font);

    SysFreeString(bstr\_text);

**if** (**SUCCEEDED**(hr))

      shape->Release();

  }

  active\_layer->Release();

**return** hr;

}

The first thing that CreateText() does is to set Application.Visible property to True. This is important for the case when CorelDRAW was not running and is automatically started by Windows when CoCreateInstance() is called. Since in this case CorelDRAW will be hidden, setting its visibility to True will make the application main window to appear on the screen.

Next we try to obtain the current document which may not exist if CorelDRAW has just been launched or if the user closed all the documents beforehand. If Application.ActiveDocument returns nullptr, we create a new document using Application.CreateDocument() method.

Finally, we create an artistic text object on the active layer of that document.

### Native C++ with helper smart pointers

The above approach has the least amount of external dependencies but the code is too verbose. Every method and invocation returns an HRESULT which needs to be validated after each call. Also, the returned interfaces are reference-counted and must be explicitly released by calling their IUnknown::Release() method. Strings need to be converted to BSTR and then freed after being used.

Fortunately, Visual Studio’s #import directive is capable of generating helper classes and methods that takes care of most of these tedious tasks. It also uses helper classes such as \_bstr\_t and \_variant\_t to abstract memory management and data conversion of BSTR and VARINT types.

Finally, the generated helper classes use exceptions instead of explicit checking for failures using HRESULTs. This makes the program much less verbose and easier to write and maintain.

To start, omit raw\_interfaces\_only and no\_smart\_pointers options for #import directive:

#import "libid:95E23C91-BC5A-49F3-8CD1-1FC515597048" version("11.5") \

      rename("GetCommandLine", "VGGetCommandLine") \

      rename("CopyFile", "VGCopyFile") \

      rename("FindWindow", "VGFindWindow")

This will still generate a header file (VGCoreAuto.tlh) with the definitions of all the CorelDRAW’s type library interfaces and objects. In addition, it will generate smart-pointer-like helper classes. For example, the main interface of the top-level application object is VGCore::IVGApplication. The compiler will also generate a helper class VGCore::IVGApplicationPtr which automatically keeps track of instance references and the underlying interface’s default methods return smart-pointers instead of raw interface pointers. While the generated .tlh file has all the declarations of the interfaces and helper classes, a new file, VGCoreAuto.tli, is also generated and contains the implementations of all the helper classes and methods. This file is compiled automatically, so there is no need to mention it anywhere in your solution.

When using the generated helper classes, the main() function body doesn’t change much. Instead of checking the return value of CreateTextInCorelDRAW(), it uses exception handling now:

**int** wmain(**int** argc, **wchar\_t**\* argv[])

{

  HRESULT hr = CoInitializeEx(**nullptr**, COINIT\_APARTMENTTHREADED);

**if** (**SUCCEEDED**(hr))

  {

**try**

    {

      CreateTextInCorelDRAW(L"Hello, world", L"Arial", 24.0f);

    }

**catch** (\_com\_error& ex)

    {

      wprintf(L"Error occurred: 0x%08X (%s)\n", ex.Error(), ex.ErrorMessage());

    }

    CoUninitialize();

  }

**return** 0;

}

As an added bonus, the exception thrown can also return the error message string in addition to just the error code.

Since there is no need to keep track of instances of objects (which is automatically handled by the smart-pointer classes), functions CreateText() and CreateTextInCorelDRAW() can be combined into one (much smaller) function:

**void** CreateTextInCorelDRAW(**const** **wchar\_t**\* text, **const** **wchar\_t**\* fontName, **float** fontSize)

{

**VGCore**::IVGApplicationPtr app(L"CorelDRAW.Application.17");

  app->Visible = **VARIANT\_TRUE**;

**VGCore**::IVGDocumentPtr doc = app->ActiveDocument;

**if** (!doc)

    doc = app->CreateDocument();

**VGCore**::IVGShapePtr shape = doc->ActiveLayer->CreateArtisticText(

    0.0, 0.0, text, **VGCore**::cdrTextLanguage::cdrLanguageMixed,

**VGCore**::cdrTextCharSet::cdrCharSetMixed, fontName, fontSize,

**VGCore**::cdrTriState::cdrUndefined, **VGCore**::cdrTriState::cdrUndefined,

**VGCore**::cdrFontLine::cdrMixedFontLine,

**VGCore**::cdrAlignment::cdrLeftAlignment);

}

Also, note the difference of how the CorelDRAW’s Application object is created. The constructor of VGCore::IVGApplicationPtr can take a CLSID or ProgID of the class to be created, so there is no need to call CLSIDFromProgID() and CoCreateInstance():

**void** CreateTextInCorelDRAW(**const** **wchar\_t**\* text, **const** **wchar\_t**\* fontName, **float** fontSize)

{

**VGCore**::IVGApplicationPtr app(L"CorelDRAW.Application.17");

  app->Visible = **VARIANT\_TRUE**;

**VGCore**::IVGDocumentPtr doc = app->ActiveDocument;

**if** (!doc)

    doc = app->CreateDocument();

**VGCore**::IVGShapePtr shape = doc->ActiveLayer->CreateArtisticText(

    0.0, 0.0, text, **VGCore**::cdrTextLanguage::cdrLanguageMixed,

**VGCore**::cdrTextCharSet::cdrCharSetMixed, fontName, fontSize,

**VGCore**::cdrTriState::cdrUndefined, **VGCore**::cdrTriState::cdrUndefined,

**VGCore**::cdrFontLine::cdrMixedFontLine,

**VGCore**::cdrAlignment::cdrLeftAlignment);

}

There are a few other important differences:

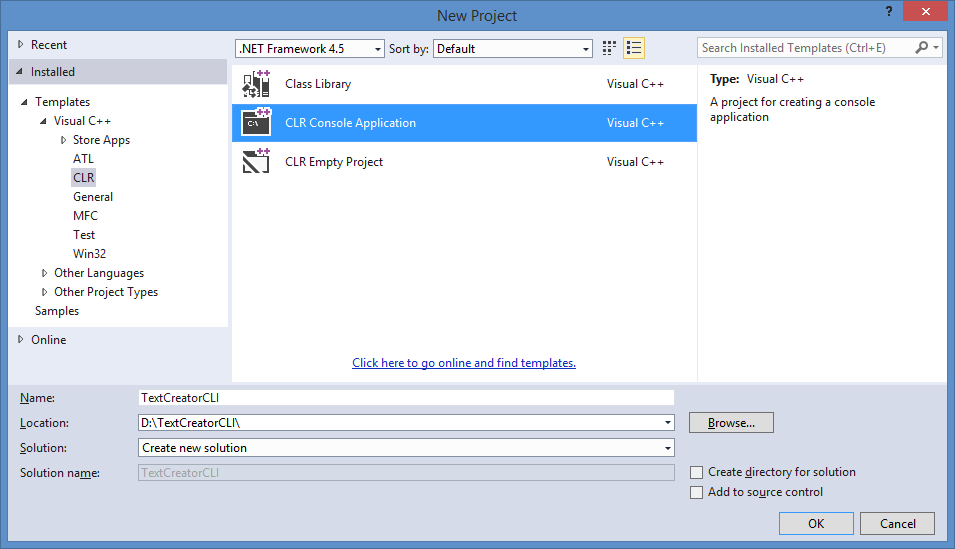
* Object properties can be just assigned values (app->Visible = … instead of app->put\_Visible(…))
* If a method or property returns another object, you can keep calling that object’s methods directly (e.g. doc->ActiveLayer->CreateArtisticText). If any step of this process fails for whatever reason, an exception will be thrown which can simple by intercepted by try…catch construct.
* None of the COM objects (app, doc, shape) is explicitly released. This automatically happens when the function exits or an exception is thrown.

## Managed .NET applications

### Managed C++/CLI

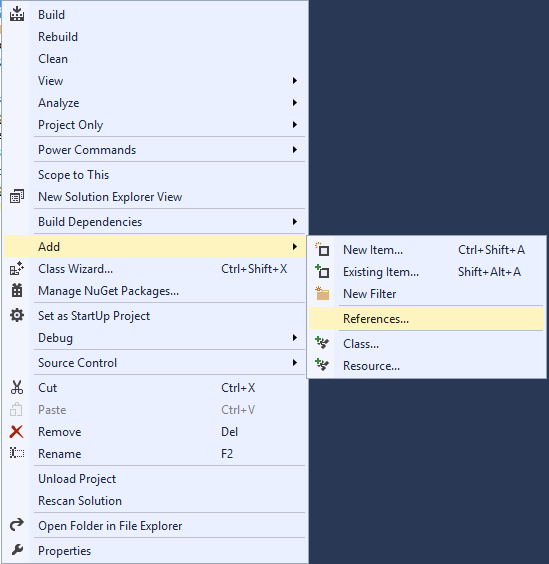
C++/CLI applications can use managed .NET assemblies, yet can mix native and managed C++ code which makes them and ideal platform for complicated solutions which need to interoperate between native and managed code.

Create a simple C++/CLI console application in Visual Studio by selecting File > New > Project… and picking “CLR Console Application” from “Visual C++ > CLR” project template category:

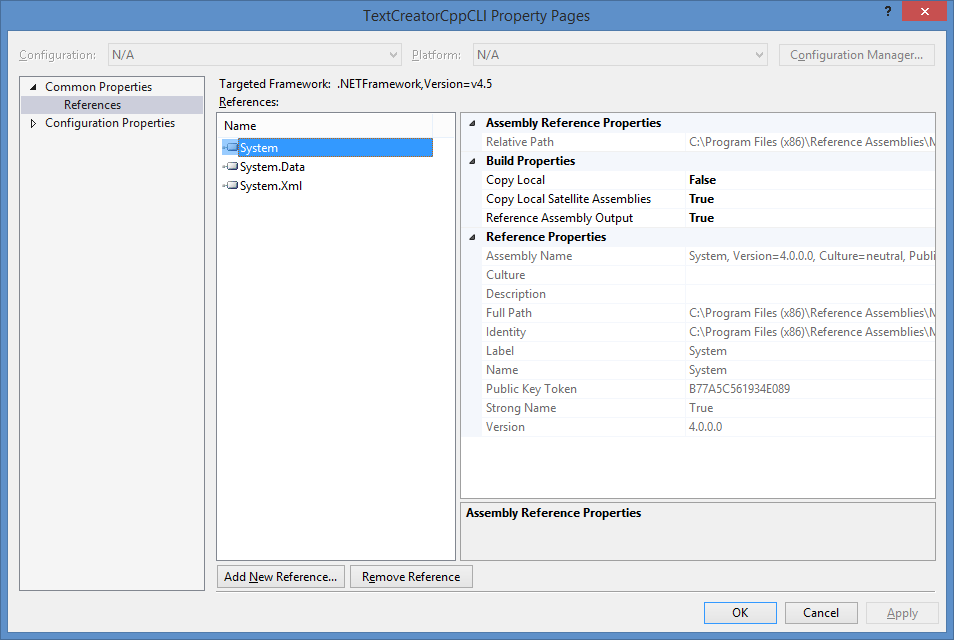


Once the project is created, you need to add a reference to CorelDRAW’s .NET assembly that is installed on the system when CDGS is installed.

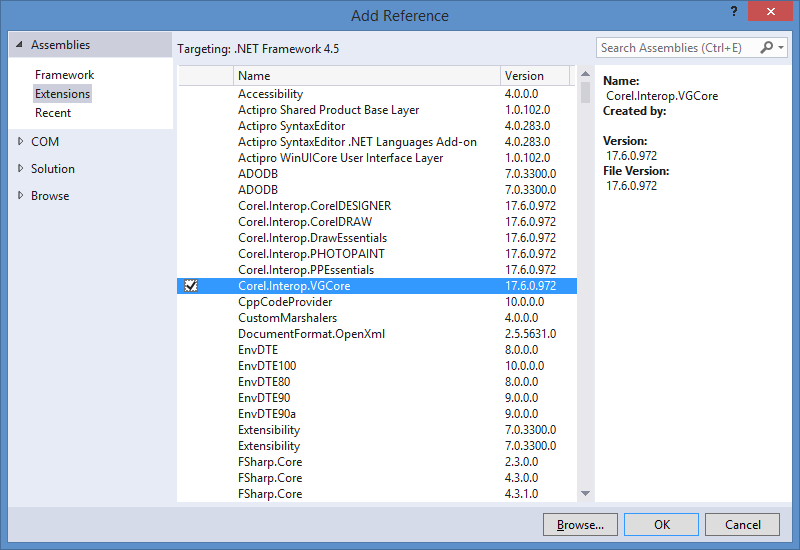
Right-click on the project node in Visual Studio’s Solution Explorer and select Add > References… or select (Project > References… from the main menu bar):



In the project property page click Add New Reference… button:



This will open Add Reference dialog. Switch to Assemblies > Extensions and pick Corel.Interop.VGCore assembly:



Once this is done, change the contents of the main .cpp file to read this:

#include "stdafx.h"

**using** **namespace** **System**;

**using** **namespace** **Corel**::**Interop**;

**void** CreateTextInCorelDRAW(**const** **wchar\_t**\* text, **const** **wchar\_t**\* fontName,

**float** fontSize)

{

  Type^ pia\_type = Type::GetTypeFromProgID("CorelDRAW.Application.17");

**VGCore**::Application^ app =  
 **static\_cast**<**VGCore**::Application^>(Activator::CreateInstance(pia\_type));

  app->Visible = **true**;

**VGCore**::Document^ doc = app->ActiveDocument;

**if** (!doc)

    doc = app->CreateDocument();

**VGCore**::Shape^ shape = doc->ActiveLayer->CreateArtisticText(

    0.0, 0.0, **gcnew** String(text), **VGCore**::cdrTextLanguage::cdrLanguageMixed,

**VGCore**::cdrTextCharSet::cdrCharSetMixed, **gcnew** String(fontName), fontSize,

**VGCore**::cdrTriState::cdrUndefined, **VGCore**::cdrTriState::cdrUndefined,

**VGCore**::cdrFontLine::cdrMixedFontLine,

**VGCore**::cdrAlignment::cdrLeftAlignment);

}

**int** main(array<**System**::String^>^ args)

{

**try**

  {

    CreateTextInCorelDRAW(L"Hello, world", L"Arial", 24.0f);

  }

**catch** (Exception^ ex)

  {

    Console::WriteLine(L"Error occurred: {0}", ex->Message);

  }

**return** 0;

}

The code above is remarkably similar to that of C++ with smart pointers, except for some C++/CLI managed code specifics.

Again, using ProgIDs here ensures that the correct application (and correct version of it) is invoked.

### Visual Basic

Just like with C++/CLI, create a new project for Visual Basic > Windows Desktop > Console Application, add a reference to Corel.Interop.VGCore assembly and in Module1.vb provide the source code for the application:

**Imports** Corel.Interop.VGCore

**Module** Module1

**Sub** CreateTextInCorelDRAW(text **As** **String**, fontName **As** **String**, fontSize **As** **Single**)

**Dim** pia\_type **As** Type = Type.GetTypeFromProgID("CorelDRAW.Application.17")

**Dim** app **As** Application = Activator.CreateInstance(pia\_type)

    app.Visible = **True**

**Dim** doc **As** Document = app.ActiveDocument

**If** doc **Is** **Nothing** **Then** doc = app.CreateDocument()

**Dim** shape **As** Shape = doc.ActiveLayer.CreateArtisticText(

      0.0, 0.0, text, cdrTextLanguage.cdrLanguageMixed,

      cdrTextCharSet.cdrCharSetMixed, fontName, fontSize,

      cdrTriState.cdrUndefined, cdrTriState.cdrUndefined,

      cdrFontLine.cdrMixedFontLine, cdrAlignment.cdrLeftAlignment)

**End** **Sub**

**Sub** Main()

**Try**

      CreateTextInCorelDRAW("Hello, world", "Arial", 24.0F)

**Catch** ex **As** Exception

      Console.WriteLine("Error occurred: {0}", ex.Message)

**End** **Try**

**End** **Sub**

**End** **Module**

Visual Basic being a little higher-level language than C++ requires even less code to do the same thing, but functionally, this example is identical to C++/CLI version.

### C#

Create a new project using Visual C# > Windows Desktop > Console Application project template and add reference to Corel.Interop.VGCore.

Apart from syntactic differences between C# and VB.NET, the program code is still very similar to that of VB.NET and C++/CLI:

**using** System.Text;

**using** System.Threading.Tasks;

**using** Corel.Interop.VGCore;

**namespace** TextCreatorCS

{

**class** Program

  {

**static** **void** CreateTextInCorelDRAW(**string** text, **string** fontName,

**float** fontSize)

    {

      Type pia\_type = Type.GetTypeFromProgID("CorelDRAW.Application.17");

      Application app = Activator.CreateInstance(pia\_type) **as** Application;

      app.Visible = **true**;

      Document doc = app.ActiveDocument;

**if** (doc == **null**)

        doc = app.CreateDocument();

      Shape shape = doc.ActiveLayer.CreateArtisticText(

        0.0, 0.0, text, cdrTextLanguage.cdrLanguageMixed,

        cdrTextCharSet.cdrCharSetMixed, fontName, fontSize,

        cdrTriState.cdrUndefined, cdrTriState.cdrUndefined,

        cdrFontLine.cdrMixedFontLine, cdrAlignment.cdrLeftAlignment);

    }

**static** **void** Main(**string**[] args)

    {

**try**

      {

        CreateTextInCorelDRAW("Hello, world", "Arial", 24.0f);

      }

**catch** (Exception ex)

      {

        Console.WriteLine("Error occurred: {0}", ex.Message);

      }

    }

  }

}

## Compiling and running the applications

No matter what the language you used to build the sample console application, you should be able to compile it with no issue and when launched, it should create an artistic text object “Hello, world” in the document, currently open in CorelDRAW. If there is no document, a new one will be created. And, finally, if CorelDRAW wasn’t already running, it will be launched automatically.

