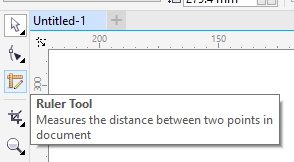
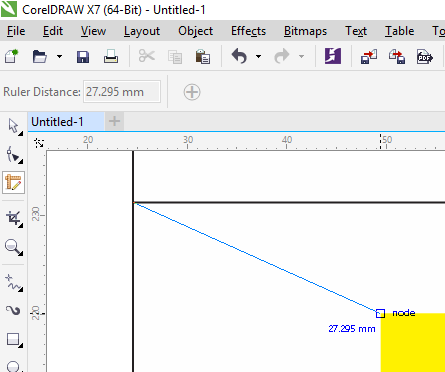
# Creating Ruler Tool for CorelDRAW X7/Corel DESIGNER X7

CorelDRAW X7 and Corel DESIGNER X7 support creation of custom tools using its plugin framework. This article describes the steps necessary for creating a simple Ruler Tool in C++. The code assumes you are using Microsoft Visual Studio 2013 or later. It is recommended to have the latest update of CorelDRAW Graphics Suite X7 installed with CorelDRAW Graphics Suite X7 Update 5 being the minimum required version.

The Ruler Tool created as part of this exercise is a simple tool that makes it possible to measure a distance between two points in a document. The sample adds a new tool to CorelDRAW/Corel DESIGNER:



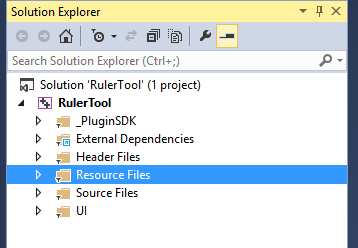
When the tool is used, it shows the distance between start/end points as the mouse is dragged over the document. The distance is displayed on the screen next to the mouse cursor as well as in the property bar:



## Opening Visual Studio Solution

Download the source code for the sample solution, extract the source code and open RulerTool.sln with Visual Studio.

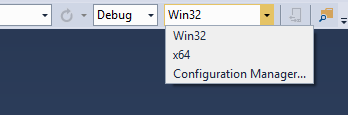
The source code is divided into a few folders inside the Solution Explorer:



* **\_PluginSDK** – is a C++ Plugin SDK source files that make creation of custom plugins for CorelDRAW/Corel DESIGENER easier. The latest source code for the SDK can be downloaded from: <http://www.oberonplace.com/cgs-plugin-sdk/>
* **External Dependencies** – dependent files outside of this project (for example, Windows Platform SDK files, etc). These files are automatically added to this folder by Visual Studio and are of no interest for the purpose of this article.
* **Header Files** – C++ header files for the solution.
* **Source Files** – C++ source files for the solution.
* **UI** – The workspace XML files and other plugin files needed for CorelDRAW/Corel DESIGNER to load the plugin DLL.

## Compiling the plugin

The source code of the solution is compiled into a CorelDRAW plugin, **RulerTool.cpg**. Note that you need to compile 32-bit or 64-bit versions (or both) of the plugin to match CorelDRAW/Corel DESIGNER version you are using. You can select the proper solution platform from the Standard toolbar of Visual Studio:



Once the correct platform is selected, hit Ctrl-Shift-B (or select Build>Build Solution menu item) in Visual Studio. This should successfully compile the code and create the plugin binary. You should see a message similar to the following in the Output window:

1> Generating Code...

1> RulerTool.vcxproj -> D:\RulerTool\Release\RulerTool.cpg

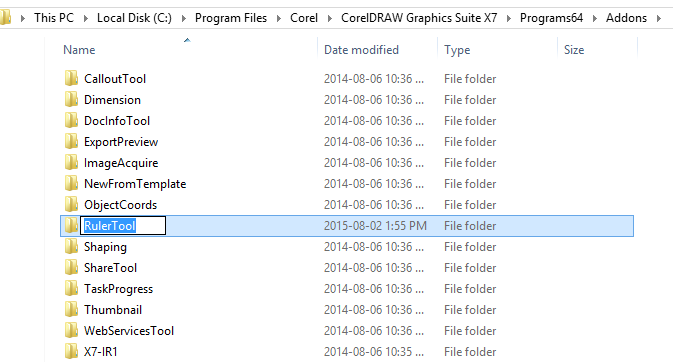
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========

## Deploying the plugin

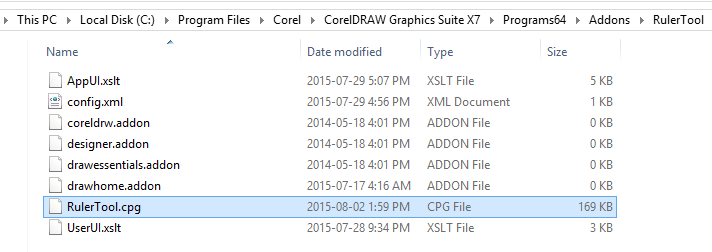
Once the plugin is compiled, you need to copy the plugin files to your CorelDRAW/Corel DESIGNER installation.

1. Create a new folder for RulerTool plugin under CDGS’s Addons folder. For example, CorelDRAW X7 version is installed in the following locations by default:
   * 64 bit - C:\Program Files\Corel\CorelDRAW Graphics Suite X7\Programs64\Addons
   * 32 bit - C:\Program Files (x86)\Corel\CorelDRAW Graphics Suite X7\Programs\Addons

Under the Addons folder, create a new folder and name it RulerTool:

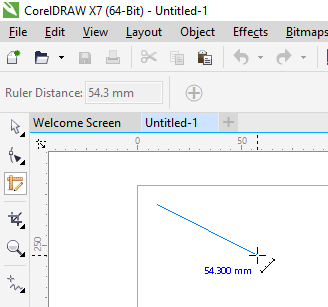


1. Copy the RulerTool.cpg plugin binary to Addons\RulerTool folder. When compiling the release version of RulerTool plugin, the resulting .cpg file will be placed in one of the following folders by Visual Studio:
   * 32 bit: …\RulerTool\Release\RulerTool.cpg
   * 64 bit: …\RulerTool\x64\Release\RulerTool.cpg
2. Copy the contents of …\RulerTool\UI folder from the sources of the Ruler Tool plugin to Addons\RulerTool inside CorelDRAW X7/Corel DESIGNER X7 installation. The contents of Addons\RulerTool should be similar to the following:



1. Start CorelDRAW/Corel DESIGNER.

When the application starts, you should notice a new tool in the Toolbox beneath the Shape Edit tool. When activated, the tool allows you to drag on the page in the active document and the distance between the original point and the current position of the mouse is shown next to the cursor as well as in the property bar:



## Looking inside the code

Open the RulerTool plugin solution in Visual Studio and look around. We are going to explore how the plugin works.

### Main plugin object

A C++ plug-in is just a DLL with .cpg extension which is automatically loaded by CorelDRAW/Corel DESIGNER at startup. The plugin DLL must export one function which is the entry point for the plugin:

**extern** "C" **\_\_declspec**(dllexport)

DWORD **APIENTRY** AttachPlugin(**VGCore**::IVGAppPlugin\*\* ppIPlugin);

This function creates an instance of the plugin object that implements IVGAppPlugin interface. This is implemented in RulerTool.cpp:

**class** MyPlugin **final** : **public** **PluginSDK**::AppPlugin {

**public**:

  // Tool state registration

**BEGIN\_TOOLSTATE\_LIST**()

**TOOLSTATE\_LIST\_ENTRY**(RulerToolState)

**END\_TOOLSTATE\_LIST**()

  // Datasources

**BEGIN\_DATASOURCE\_LIST**()

**DATASOURCE\_LIST\_ENTRY2**("RulerToolDS", RulerToolDataSource)

**END\_DATASOURCE\_LIST**()

**private**:

  HRESULT LoadTypeLib(ITypeLib\*\* pTypeLib) **override**;

};

**DECLARE\_APP\_PLUGIN**(MyPlugin)

This plugin object uses helper class PluginSDK::AppPlugin from the Plugin SDK and removes some of the complexities of implementing IVGAppPlugin interface. After the plugin class is declared and implemented, it is exported to CorelDRAW via DECLARE\_APP\_PLUGIN() macro which is defined as follows in \_PluginSDK\AppPlugin.h:

#define **DECLARE\_APP\_PLUGIN**(ClassName) \

**extern** "C" **\_\_declspec**(dllexport) \

DWORD **APIENTRY** AttachPlugin(**VGCore**::IVGAppPlugin\*\* ppIPlugin) { \

    \*ppIPlugin = **new** ClassName; \

**return** 0x100; \

  }

It simply declares the AttachPlugin method and creates an instance of MyPlugin when called by the host application.

The plugin class provides a list of tool states implemented in the plugin (in our case, just one – RulerToolState) and datasources used by user interface controls. We declare one datasource, RulerToolDataSource, which exposes the current measured length to the control on the property bar.

### Tool States

Interactive tools in CorelDRAW/Corel DESIGNER define state objects responsible of handling user input. When the application enters a particular state, the corresponding state object handles all the mouse and keyboard input. CorelDRAW has a number of internal states that it uses when working with a document and developers can provide their own additional states to help implement new tools.

A state must implement IVGToolState interface which delivers mouse and keyboard input as well as provide basic tool set for working with input data (e.g. to perform mouse snapping and constraints). Plugin SDK wrapper classes, such as PluginSDK::ToolState<T>, simplify implementation of IVGToolState interface.

The Ruler Tool state object overrides some of those methods to implement the necessary interactions with the host application:

**class** **\_\_declspec**(uuid("a5bdc561-e254-a088-4c94-ed1eb6f8e320")) RulerToolState

    : **public** **PluginSDK**::ToolState<RulerToolState> {

**public**:

  RulerToolState(**PluginSDK**::AppPlugin\* appPlugin);

  ~RulerToolState();

**void** OnStartState() **override**;

**void** OnMouseMove(**VGCore**::IVGPointPtr pt) **override**;

**void** OnLButtonDownLeaveGrace(**VGCore**::IVGPointPtr pt) **override**;

**void** OnLButtonUp(**VGCore**::IVGPointPtr pt) **override**;

**void** OnAbort() **override**;

**void** OnCommit(**VGCore**::IVGPointPtr pt) **override**;

**bool** OnSnapMouse(**VGCore**::IVGPointPtr pt) **override**;

**private**:

**VGCore**::IVGPointPtr m\_StartPoint;

**VGCore**::IVGOnScreenCurvePtr m\_Preview;

**VGCore**::IVGOnScreenTextPtr m\_PreviewText;

};

Note that the UUID associated with RulerToolState class will be used to register this state class with the host application. This ID must be unique and specific for each object. When creating your own states, make sure to generate new UUIDs for them.

When the Ruler Tool is activated, the RulerToolState object is created and OnStartState() method is invoked. This method is setting some default state behaviours for the tool:

**void** RulerToolState::OnStartState() {

  m\_stateAttributes->SetCursorGuid("a5bdc561-e254-a088-4c94-ed1eb6f8e320");

  m\_stateAttributes->AllowTempPickState = **VARIANT\_FALSE**;

  m\_stateAttributes->PropertyBarGuid = "07da4f59-2cfd-4d8a-49bc-6d121ebcd47b";

  m\_Preview = GetHostApplication()->CreateOnScreenCurve();

  m\_PreviewText = GetHostApplication()->CreateOnScreenText();

}

It uses m\_stateAttributes (of type IVGToolStateAttributes) to sepecify the default cursor used for the tool by calling SetCursorGuid() method. The GUID used for the cursor identifier is mapped to the .cur cursor file embedded in the resources of the plugin. See the section on Resource Mapping below for more details.

AllowTempPickState controls whether the selection handles around selected object(s) are to be shown while the tool is active and whether the Pick tool can be activated when the cursor is over any of the handles, allowing the tool to resize and move objects without exiting the tool. Since the ruler tool does not need this functionality, it is disabled by assigning VARIANT\_FALSE to that property.

PropertyBarGuid specifies the GUID of the property bar mode to be shown when the tool is active. Since this plugin provides its own property bar when in ruler tool mode, the ID of that toolbar is specified (see User Interface section for more details).

Finally, the function creates the on-screen curve and text to be used for the preview of the ruler and display the measured length right in the document.

When the left mouse button is pressed, the application temporary enters a Grace Move state which means that you need to move the mouse more than a few pixels to detect a mouse move event. This helps to distinguish single click events and click-and-drag (since the mouse could move a little while clicking the mouse button). This behaviour is controlled by EnterGraceStateOnLButtonDown property of m\_stateAttributes which is enabled by default.

When a Grace Move state exits because the mouse was moved far enough, it is treated as click-and-drag event and the tool state receives OnLButtonDownLeaveGrace notification with the position of the original mouse-down event.

**void** RulerToolState::OnLButtonDownLeaveGrace(**VGCore**::IVGPointPtr pt) {

  m\_StartPoint = pt;

  m\_drawing = **true**;

**auto** ds = m\_appPlugin->GetDataSource<RulerToolDataSource>();

**if** (ds)

    ds->StartMeasuring();

}

In this method, we remember the point of original click so we can use it to measure the distance from it to the current mouse position. We also use the RulerToolDataSource to update the property bar (so the distance control starts showing the distance value, as opposed to being empty).

While dragging the mouse, we want the cursor to snap to objects in the document and be able to constrain the mouse movement using the constraint keys Shift/Ctrl. This is achieved by responding to OnSnapMouse() method in the state. The original mouse position is passed in as an argument and the function is free to modify it as necessary:

**bool** RulerToolState::OnSnapMouse(**VGCore**::IVGPointPtr pt) {

**if** (m\_drawing) {

    m\_stateAttributes->AnchoredSnapMouse(pt, m\_StartPoint);

    m\_stateAttributes->ConstrainMouse(pt, m\_StartPoint);

**return** **true**;

  }

**return** **false**;

}

In our case we call AnchoredSnapMouse and ConstrainMouse methods of m\_stateAttributes object to perform snapping and constraining respectively. But we do this only while we are actually moving the mouse with the left mouse button depressed (m\_drawing).

Finally, after the mouse position was modified during snapping/constraint, it is being passed to the tool state object in OnMouseMove() event. A simplified version of this event handler is shown below. The actual implementation is a bit more complicated because it takes care of having the text label not to overlap the mouse cursor and updating the property bar:

**void** RulerToolState::OnMouseMove(**VGCore**::IVGPointPtr pt) {

**if** (m\_drawing) {

    m\_Preview->SetLine(m\_StartPoint->x, m\_StartPoint->y, pt->x, pt->y);

**auto** doc = GetHostApplication()->ActiveDocument;

**double** length = m\_StartPoint->DistanceTo(pt);

**double** ui\_distance = doc->FromUnits(length, doc->Rulers->HUnits);

    CStringW text;

    text.Format(L"%.3f %s", ui\_distance, GetUnitName(doc->Rulers->HUnits));

    m\_PreviewText->SetTextAndPosition(text.GetString(), pt->x, pt->y, align, 0, 0);

    m\_Preview->Show();

    m\_PreviewText->Show();

  }

}

Here we update the preview curve to be a line between the original click position (m\_StartPoint) and the current mouse cursor position in the document (pt). Then we calculate the distance between those two points and convert the value to the document units (e.g. millimeters or inches) and show them as a text label next to the mouse cursor.

Finally, when the left mouse button is released, OnLButtonUp event is generated in which case we just remove the on-screen information:

**void** RulerToolState::OnLButtonUp(**VGCore**::IVGPointPtr pt) {

  m\_Preview->Hide();

  m\_PreviewText->Hide();

  m\_drawing = **false**;

}

When Enter key is pressed, OnCommit event is generated which Esc key generated OnAbort event. For the sake of this example we do the same as releasing the mouse button – hiding the on-screen information.

### Datasources

When the plugin needs to interact with the user interface of CorelDRAW/Corel DESIGNER, it registers one or several datasource objects which the user interface controls can bind to in order to retrieve the necessary information to display in the UI.

The Ruler Tool provides a datasource for the property bar control to display the current distance measured and a Boolean to specify whether we are currently measuring distance. This allows the property bar control to display the distance value if we are measuring, or stay blank/empty if we are not.

Datasource objects are COM object that implement IDispatch interface and expose methods and/or properties that UI controls can bind to. To help with implementation of type reflection and method invocation, the actual datasource interface should be described in an IDL file and compiled into a type library. Our example datasource has the following interface:

[**object**, **uuid**(2AC7496C-1F25-4D57-9C96-9BB922ED1830), **dual**]

**interface** IRulerDataSource : IDispatch {

  [**propget**, **id**(1)] HRESULT Distance([**out**, **retval**] **double**\* pVal);

  [**propget**, **id**(2)] HRESULT IsMeasuring([**out**, **retval**] VARIANT\_BOOL\* pVal);

};

The actual implementation is provided by RulerToolDataSource object:

**class** RulerToolDataSource

 : **public** **PluginSDK**::DataSource<RulerToolDataSource, IRulerDataSource> {

**public**:

  RulerToolDataSource(CComPtr<ITypeLib> typeLib,

**VGCore**::ICUIDataSourceProxyPtr proxy)

      : DataSource{typeLib, proxy} {}

**void** SetDistance(**double** distance);

**void** StartMeasuring();

**void** StopMeasuring();

**private**:

**STDMETHOD**(get\_Distance)(**double**\* pVal) **override**;

**STDMETHOD**(get\_IsMeasuring)(VARIANT\_BOOL\* pVal) **override**;

**bool** m\_measuring{**false**};

**double** m\_distance{0.0};

};

When a datasource property is changed, the UI framework needs to be notified of the change so that the UI controls can update their values. This is done by calling UpdateListeners method on the datasource proxy class that is provided to the instance of the datasource object when it is created:

**void** RulerToolDataSource::SetDistance(**double** distance) {

  m\_distance = distance;

  m\_proxy->UpdateListeners("Distance");

}

**void** RulerToolDataSource::StartMeasuring() {

  m\_distance = 0.0;

  m\_measuring = **true**;

  m\_proxy->UpdateListeners("Distance,IsMeasuring");

}

**void** RulerToolDataSource::StopMeasuring() {

  m\_measuring = **false**;

  m\_proxy->UpdateListeners("IsMeasuring");

}

Here each time we update a value of a property, we make sure that value is reflected in the UI by updating the list of properties that are actually changed.

### User interface bindings

User interface elements are described in XML and are loaded by CorelDRAW/Corel DESIGNER at start up. The UI is described by first defining individual UI Items (controls) with control properties bound to certain datasource properties. For example, the Ruler Distance control from the ruler tool plugin looks like this:



And it is defined in UI\AppUI.xslt as follows:

<!-- Ruler Tool distance control in property bar -->

<itemData guid="01e04e8a-b8b2-88a2-4d1a-14f4184c980d"

          icon="guid://496ea244-5a15-4d15-bb7c-4cb1ee27db96"

          type="spinner"

          showSpinner="false"

          showAmbiguous="\*Not(\*Bind(DataSource=RulerToolDS;Path=IsMeasuring))"

          value="\*Bind(DataSource=RulerToolDS;Path=Distance)"

          leftLabelText="\*ResS('1e2247a4-3b70-b5a3-4fb3-109d467ed64e')"

          enable="false"

          numDecimalPlaces="\*Bind(DataSource=WPrefsApp;Path=DrawingPrecision)"

          increment="0.1 in; 1 mm; 1pt; 1px"

          rangeMin="0"

          rangeMax="\*Bind(DataSource=WPickDataSource;Path=PositionXMax)"

          displayUnit="\*Bind(DataSource=DocumentPrefsDS;Path=XRulerUnit)"

          showUnit="true"

          customizationMode="4fbdcedf-c475-4719-91f2-beb193169730"

          scale="\*Bind(DataSource=DocumentPrefsDS;Path=WorldScale)"

          />

This definition describes a spinner control (type=”spinner”) with unique identifier of “01e04e8a-b8b2-88a2-4d1a-14f4184c980d”.

The value displayed in the spinner comes from the property “Distance” of datasource “RulerToolDS”.

It also specifies some other bindings like the current document units (displayUnit), document drawing scale (scale) which are provided by standard CDGS datasource DocumentPrefsDS.

AppUI.xslt along with UserUI.xslt also describe toolbars, menus and dialogs that the plugin may display. In our case, we define a new property bar mode that contains just our spinner control displaying the ruler distance:

<!-- Ruler Tool property bar mode -->

<modeData guid="07da4f59-2cfd-4d8a-49bc-6d121ebcd47b">

   <item guidRef="01e04e8a-b8b2-88a2-4d1a-14f4184c980d"/>

</modeData>

Here we define the property bar mode ID as “07da4f59-2cfd-4d8a-49bc-6d121ebcd47b” (note this ID being used in RulerToolState::OnStartState() when setting it to PropertyBarGuid) and place a single control, with ID of “01e04e8a-b8b2-88a2-4d1a-14f4184c980d”, on it.

The UI files also describe the actual tool icon shown in the toolbox:

<!-- Ruler Tool item in the Toolbox-->

<itemData guid="a5bdc561-e254-a088-4c94-ed1eb6f8e320"

          type="groupedRadioButton"

          currentActiveOfGroup="\*Bind(DataSource=WAppDataSource;Path=ActiveTool;BindType=TwoWay)"

          enable="\*Bind(DataSource=WAppDataSource;Path=ToolboxEnable)"

          identifier="a5bdc561-e254-a088-4c94-ed1eb6f8e320"

          customizationMode="28295234-2c3b-4977-80c6-c2dbbd14fe0a"/>

This is a rather standard definition of a toolbox item which is defined as “groupedRadioButton” (because only one tool button can be active at a time) and its “currentActiveOfGroup” property is bound to WAppDataSource::ActiveTool binding provided by the host application which automatically activates the state with the ID specified in “identifier” property.

Note that our tool state object has the same ID associated with the class:

**class** **\_\_declspec**(uuid("a5bdc561-e254-a088-4c94-ed1eb6f8e320")) RulerToolState;

### Resource mapping

You may have noticed that the XML file doesn’t contain text strings or icons directly but rather has some GUID values for them. The way string and icon resources are created and used in plugins is as follows:

The string and image resources are embedded into the resource section of .cpg file the same way the resources are embedded into any other DLL. The resource file (RulerTool.rc) has the definitions of those resources:

STRINGTABLE

BEGIN

  IDS\_STRING100 "\001CT=Ruler Tool\001TH=Ruler Tool\001TT=Measures the distance "

                "between two points in document"

  IDS\_STRING101 "Ruler Tool"

  IDS\_STRING102 "Distance"

  IDS\_STRING103 "Ruler Distance:"

END

IDI\_RULERTOOL  ICON   "RulerTool.ico"

IDC\_RULERTOOL  CURSOR "RulerTool.cur"

And the IDs of those resources are provided in Resource.h file:

#define **IDS\_STRING100**  100

#define **IDS\_STRING101**  101

#define **IDS\_STRING102**  102

#define **IDS\_STRING103**  103

#define **IDI\_RULERTOOL**  200

#define **IDC\_RULERTOOL**  200

So, the first string has a string ID of 100, the second one (“Ruler Tool”) has ID of 101 and so on.

Finally, the mapping between native resource IDs and corresponding GUIDs from the XML files is provided by UI\config.xml:

<?xml version="1.0"?>

<config>

  <resources>

    <resource name="RulerToolResources" path="RulerTool.cpg">

      <resourceMap>

        <resEntry id="a5bdc561-e254-a088-4c94-ed1eb6f8e320" string="100"

 icon="200" cursor="200"/>

        <resEntry id="07da4f59-2cfd-4d8a-49bc-6d121ebcd47b" string="101"/>

        <resEntry id="01e04e8a-b8b2-88a2-4d1a-14f4184c980d" string="102"/>

        <resEntry id="1e2247a4-3b70-b5a3-4fb3-109d467ed64e" string="103"/>

      </resourceMap>

    </resource>

  </resources>

</config>

This file tells the host application that the resources for this plugin should be loaded from “RulerTool.cpg” and GUID of “a5bdc561-e254-a088-4c94-ed1eb6f8e320” corresponds to the string with ID 100, icon with ID 200 and cursor with ID of 200. This GUID is also used for the ID of the toolbox button for the ruler tool, so that button automatically gets the correct icon and caption/tooltip strings. OnStartState() function also calls “m\_stateAttributes->SetCursorGuid("a5bdc561-e254-a088-4c94-ed1eb6f8e320")” which is where the tool cursor shape comes from.

The intricacies of UI definition language and databinding used in CorelDRAW Graphics Suite are outside of the scope of this article and deserve a separate topic on its own. But you can look for examples of control definitions used by CorelDRAW itself by looking at its UI definition file which is normally installed in “C:\Program Files\Corel\CorelDRAW Graphics Suite X7\Draw\UIConfig\DrawUI.xml”. Just make sure not to modify this file or you risk damaging your CDGS installation beyond repair.