An Experiment in Visual Programming

Doug Blank

CS110 Fall 2008

VPL Tutorial 1 - Hello World

This tutorial is the classic introduction for most programming languages that displays the words "Hello World". This is easily accomplished in Microsoft Visual Programming Language (VPL) using two activity blocks, a Data block and the Simple Dialog block.

This tutorial is provided in the Microsoft Visual Programming Language (VPL) language. You can find the project files for this tutorial at the following location under the installation folder:

Create a Hello Word Diagram in VPL

To run VPL, from the **Start** menu choose **All Programs** and then click on **Visual Programming Language** under the **Microsoft Robotics Developer Studio** installation folder.

From the **File** menu click **New** to create a new project. Insert a **Data** activity, by double-clicking on the icon or draggging and dropping it from the **Basic Activities** toolbox. Choose **string** from the drop-down list. Click in the text box of the **Data** activity block and type **Hello World**.

Figure 1 - Data Activity Block

Insert a **Simple Dialog** activity block by dragging one from the **Services** toolbox and place it to the right of the **Data** activity block. (To save time looking for a service, you can type the name of the service you are looking for into the top of the **Services** toolbox. The toolbox will then display any matching activities.)

Drag a link starting from the output connection pin of the **Data** activity block onto the **Simple Dialog** activity block. The **Connections** dialog box automatically opens. Choose **DataValue** in the first list and **AlertDialog** in the second list, then click **OK**.

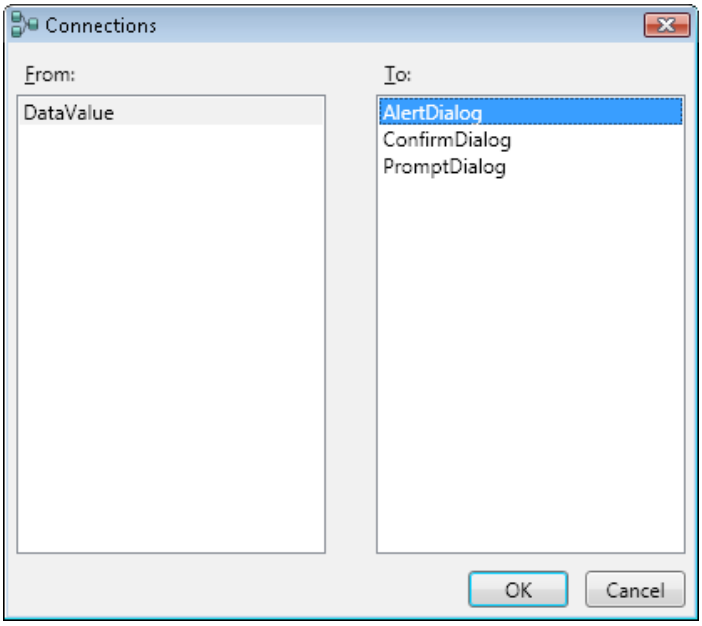

Figure 2 - Connections Dialog Box

The **Data Connections** dialog box opens next. In the drop-down list, choose **value**.

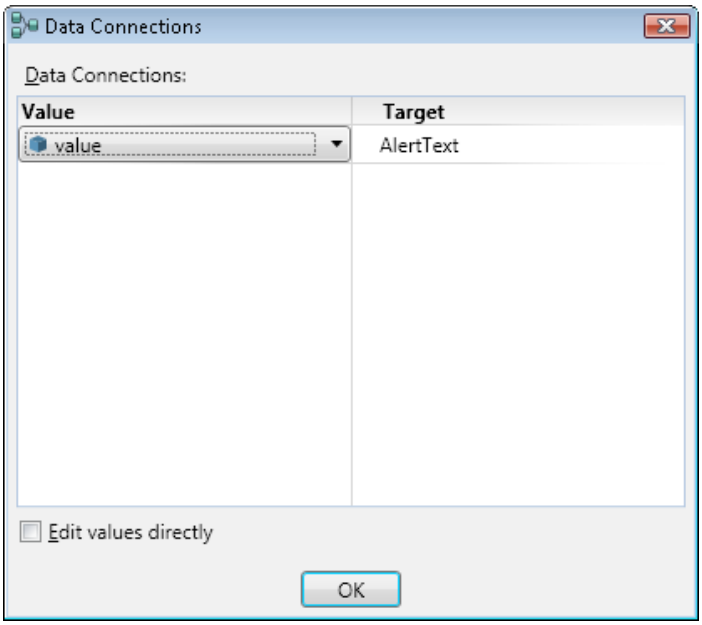

Figure 3 - Data Connections Dialog Box

The **Data Connections** dialog box tells VPL that you want to apply the value of the **Data** activity to the message text for the Alert form of this dialog.

Your diagram should now look like the following.

Figure 4 - Completed Diagram

Now choose the **Run** command from the **Run** menu (or press **F5**). If you have not saved your project yet, VPL opens the Save dialog. Enter a name for your project and click **Save**.

VPL should now run your application. If you get a message asking whether to unblock the application, choose **Unblock**.

A simple **Alert** dialog box should appear with the text **Hello World** in it.

Figure 5 - Alert Dialog

To stop the application, click the **Stop** button in the Run dialog.

VPL Tutorial 2 - Incrementing a Value

This tutorial introduces additional basic **Dataflow Control** activities included in VPL. You will create a variable; initialize it; and, using **Text-To-Speech**, audibly count to ten.

This tutorial is provided in the Microsoft Visual Programming Language (VPL) language. You can find the project files for this tutorial at the following location under the installation folder:

Create and set a Variable

To begin, create a new project by choosing **New** from the **File** menu. Next, insert a **Variable** activity by dragging it into the tool box or double-clicking it.

Click the Ellipsis Button (**…**) on the **Variable** activity block opening the **Define Variables** dialog. (You can also select "Variables..." from the **Edit** menu). In the dialog box, click on the **Add** button and type **Test** in the text box. In the **Type** drop-down list make sure **int** is selected as the variable type. Click **OK**.

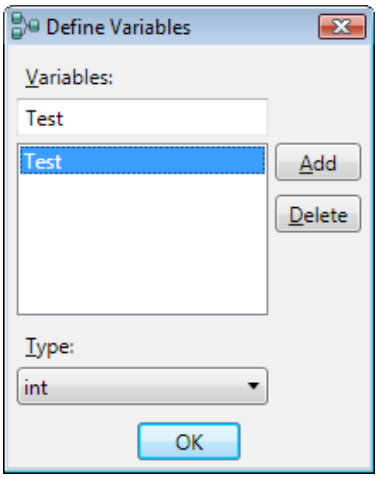

Figure 1 - Define Variables Dialog

If your **Test** variable does not show as the current setting in the **Variable** text box, open the drop-down list again and select your newly created variable to assign it to this activity.

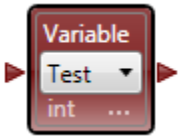

Figure 2 - Variable Activity

Add a **Data** block to your diagram to the left of the Variable block and connect them by dragging a link connection from the output pin of the **Data** activity block to the **Variable** activity block. The **Connections** dialog box opens. Select **DataValue** for the outgoing connection pin and **SetValue** for the incoming connection pin and click **OK**.

Figure 3 - Data Block setting a Variable

Select **int** from the drop-down list and enter **1** into the text box of the **Data** activity block. This sets both the data and its type. Its connection will then initialize the **Test** variable to **1**. When you

use the **SetValue** connection of a **Variable** activity block, it not only sets **1** as the value, but also sends the variable on through its output connection.

Test the value of a variable

Add a **Merge** activity block to the right of your **Variable** activity block and connect the **Variable** activity block to the **Merge** activity block. You will use this block to create a counting loop. A **Merge** activity block can have multiple inputs, each input is passed on as it is received.

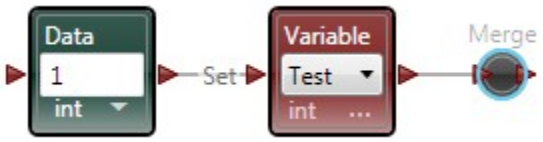

Figure 4 - Merge Block

Add an **If** activity block to the diagram to right of the **Merge** activity block. Connect the **Merge** activity block outgoing connection to the **If** activity block. In the **If** activity block, enter **Test == 10** in the test condition. (The syntax with a double equal sign is from C#. You can also use a single equal sign in VPL.) This causes the **If** activity block to check if the variable **Test** is equal to **10**, and send a message via its normal output pin if it is, or send a message through the **Else** pin if it is not equal.

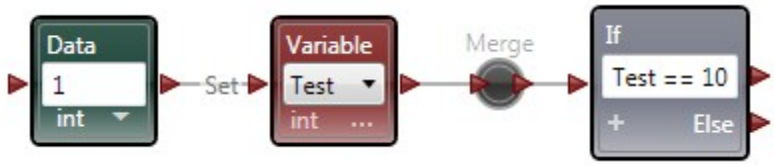

Figure 5 - If Block

Increment a variable

Add a **Calculate** activity block. Right-click on the **Calculate** block and select "Flip Connections" from the pop-up context menu. This swaps the input and output pins so that the connections on the diagram flow in a more natural way. Connect it to the **Else** connection (the lower output pin) of the **If** activity block. This connection is used if the test condition is false, i.e. **Test** is not equal to **10**. In the **Calculate** activity block, type **Test + 1**.

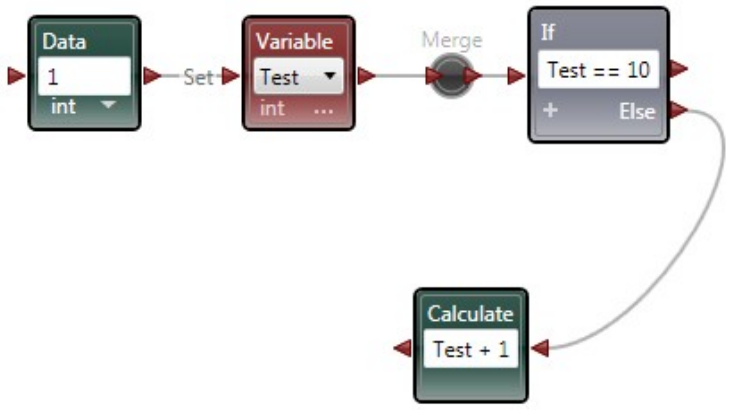

Figure 6 - Increment a Variable

The **Calculate** takes the value of **Test** and adds **1** to it then sends the result through its output pin.

Use this result to update the value of **Test** by inserting another **Variable** activity block (set it to **Test** using its drop-down list) and connecting it to the output of the **Calculate** box. The **Connections** dialog box should open. Select **CalculatedResult** and **SetValue** and click **OK**. Connect the output connection pin of the **Variable** activity block to the **Merge** activity block. This completes your loop.

When you make the final connection of the loop to the **Merge**, an exclamation icon will appear. If you hover over this with the cursor you will see a warning message "Loop detected. Consider using recursion!". This example does not use recursion, and you can ignore this warning because it is not relevant.

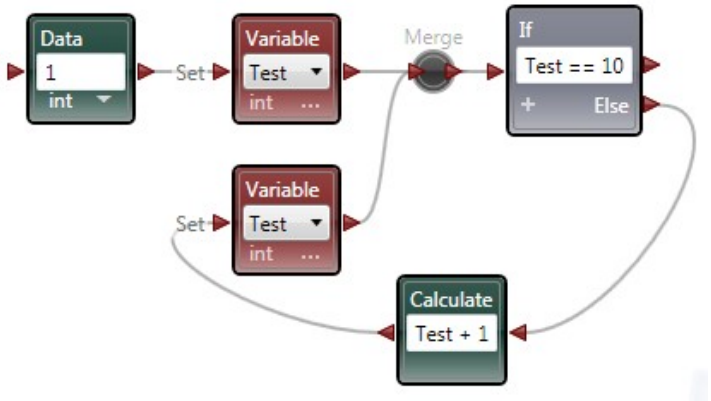

Figure 7 - Closing the Loop

Use Text To Speech

To make things a little more interesting, add another **Calculate** activity block and connect it to the output of the **Merge** activity block. Type **"The number is " + Test** into the **Calculate**. This

automatically converts the value of **Test**to text and appends it to the words inside the quotation marks.

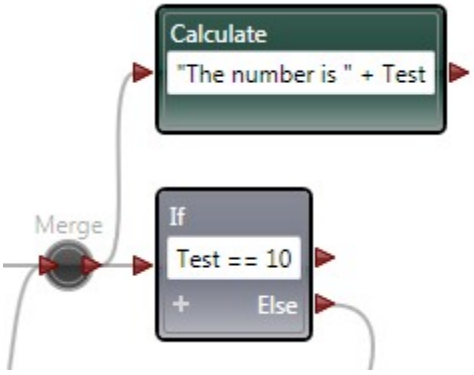

Figure 8 - Creating a Text Message

Add a **Text-To-Speech** service activity block and connect it to the output of the **Calculate** activity block. Set the Connections from the **CalculatedResult** to the **SayText** and the Data Connections from **value** to **SpeechText**. This results in the text-to-speech engine speaking on each iteration of the loop.

In this example you use **SayText**. However, there is also a **SayTextSynchronous** operation. The difference is that **SayTextSynchronous** delays sending a response until the service has finished speaking. This might be important for some applications. In this case though, the loop runs 10 times and queues up a series of speech messages. If you watch the diagram in the debugger (running at full speed) you will see that it finishes long before the computer finishes saying all of the messages. Saying the messages is not done inside the loop, so it does not slow down the loop.

Figure 9 - Say Text

To finish the tutorial, insert another **Data** activity block and connect it to the top output connection of the **If** activity block. Choose **string** from the drop-down list and type **All done!** into the text box of the **Data** activity block. (Notice that when you are defining string data you do not have to enclose it in quotation marks). Now add another **TextToSpeech** activity block (you can copy the existing one) and connect to the **Data** activity block. Select From: **DataValue**, To: **SayText** in the **Connections** dialog box. In the **Data Connections** dialog, select **value** to map to **SpeechText**.

Figure 10 - Say Message when Finished

Your finished diagram should look like the following illustration.

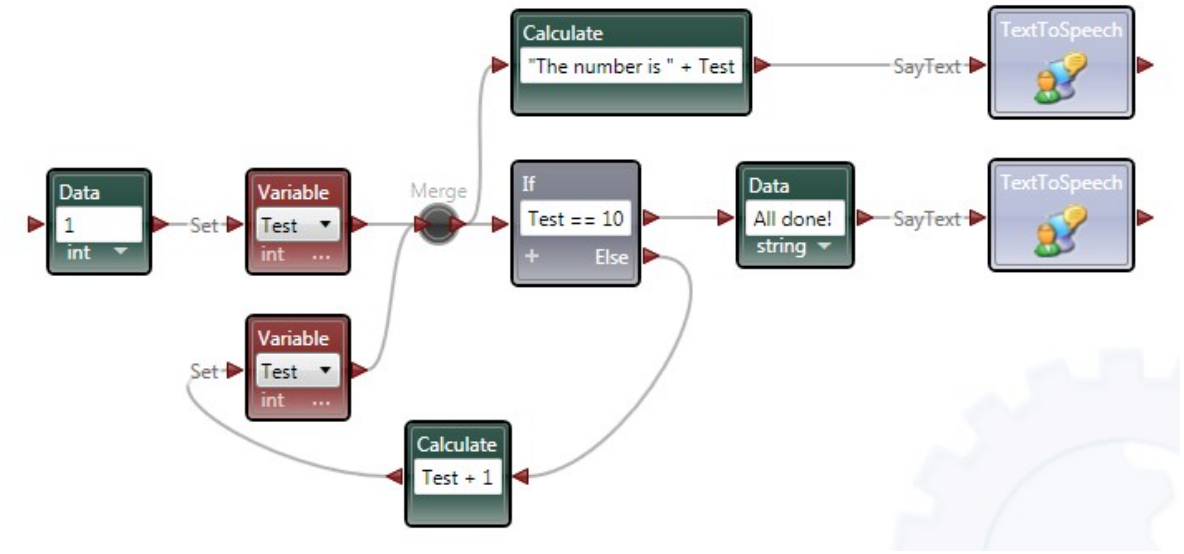

Figure 11 - Completed Diagram

Run the Program

Now, if you connected everything up correctly, you should be able to Run the application (by selecting the **Start** command on the **Run** menu or pressing **F5**). You will hear your PC count to ten and then say "All done". If you don't hear anything, check your connections and the volume for your speakers. (If you don't have speakers you could use the Simple Dialog service used in VPL Tutorial 1 - Hello World in place of the **TextToSpeech** service.)

VPL Tutorial 3 - Create Your Own Activity

This tutorial uses the same scenario as VPL Tutorial 2 - Incrementing a Value, but illustrates how you can create your own activities to modularize your application.

This tutorial is provided in the Microsoft Visual Programming Language (VPL) language. You can find the project files for this tutorial at the following location under the installation folder:

Create an Activity

Create a new project (using the **File** > **New** menu command) and insert a **Data** activity. Set the value of your new **Data** activity block to **1**.

Create a new activity by inserting an **Activity** block on your diagram. Click its title and rename it **CountTo10**, then open it. (This is easily accomplished by selecting the **Open** command on its pop-up context menu or double-clicking it). A new tabbed page should appear. Using the **Action** drop-down list at the top of the pag, click the **Action** entry. This switches to the **Action** handler page. While the diagram page looks similar to your top level data-flow page, you can see that it has connection borders on each side which is how you connect its internal data-flow to an external data-flow.

Use the **Actions and Notifications** command in the **Edit** menu or click the button next to the action selector drop-down list at the top of the page. Click the **Action** in the list. Add an input value to the action. This is the start value that you will use later for counting. Change the name of the input value to **StartValue** and choose **int** as its type. Click **OK**.

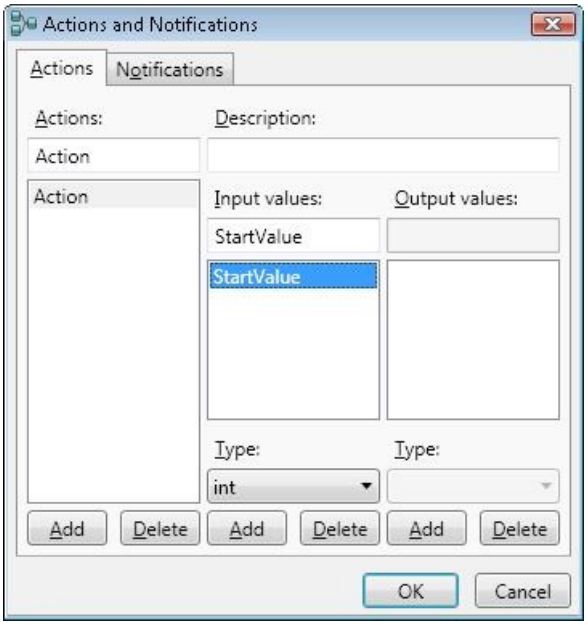

Figure 1 - Actions and Notifications Dialog

Insert the activity blocks you used in [VPL Tutorial 2 - Incrementing a Value] as part of the **Activity**. These activities were **Merge**, **If**, **Calculate**, **Variable**, and **Data**. However, you will need to connect them slightly differently to make them work in this tutorial.

Begin by placing a **Calculate** activity block near the left border and connect the action's input pin. Type **StartValue** in the **Calculate** textbox. This extracts the start value from the action's input message.

Place a **Merge** activity block next to the **Calculate** activity block and connect them. Place the **Variable** activity block to the right of the **Merge** activity block. Use the incoming value that comes into the activity routed through the **Merge** activity block to set the variable you wish to use.

Create a new variable and name it **Test2**. To create a variable, display the drop-down list in the **Variable** activity block and click **Define Variables...**. In the resulting dialog box, enter the name and type for the variable and set its type to **int**. Connect the output of the **Merge** activity block to the **Variable** activity block. This causes the value coming through the **Merge** activity block to set the value of the **Variable** activity to **Test2**.

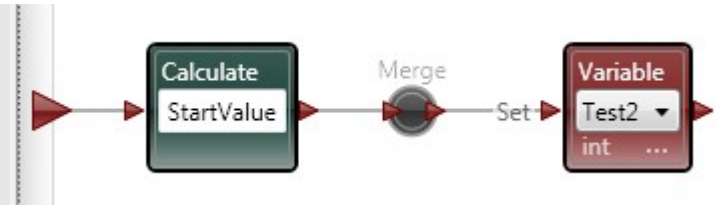

Figure 2 - Set a Variable

Connect the output of the **Variable** activity block (Current Value) to the **If** activity block (Condition). In the **If** text box, type **Test2 = 10**. This test checks the limit on the number of times to execute.

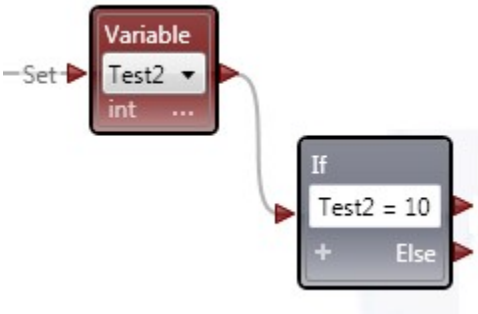

Figure 3 - Add If Activity

Create a new **Calculate** activity. Use the pop-up context menu of the **Calculate** activity (right-click on the block) to flip the connections so that the input enters on the right instead of the left.

Connect the output of the **Else** connection on **If** activity block to the input of a **Calculate** activity block. In the **Calculate** text box, type **Test2 + 1** to add one to the current value of **Test2**. Connect the output of the **Calculate** activity block to the input of the **Merge** activity block. (A

Merge accepts multiple incoming connections). This updates the value of the variable when the message flows through to the **Variable** activity. Notice that this creates a loop.

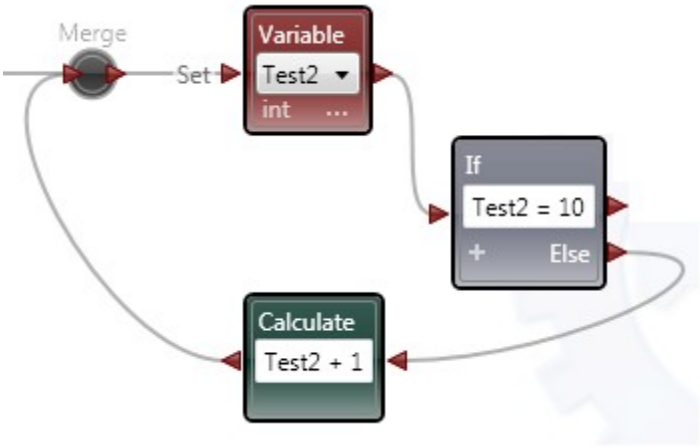

Figure 4 - Creating a Loop

Now, like in VPL Tutorial 2 - Incrementing a Value, connect a **Calculate** activity block to the output of the **Variable** activity block. Type **"The number is " + Test2** in the **Calculate** activity block's text box.

Insert and connect a **Data** activity block to the top output of the **If** activity block. Select **string** from the drop-down list, and type **All done!** in the text box. Use a second **Merge** activity block to connect the output of the **Calculate** activity block and **Data** activity block. This simplifies the connection to other data-flows outside this action.

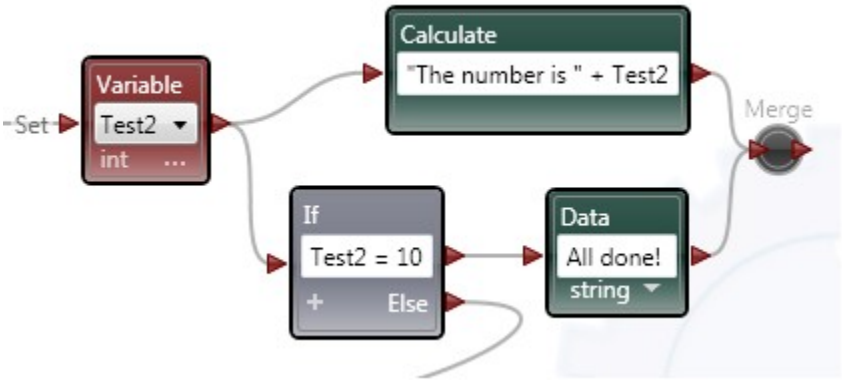

Figure 5 - Finish adding activities

To connect the output of your data-flow, select the **Actions and Notifications** command from the **Edit** menu. In the dialog box, click the **Notifications** tab and then click the **Add** button under Notifications to create a new notification. Then click on **Add** under Notification values. Name the new value **Text** by clicking **Add** and set the type as **string**, then click **OK**.

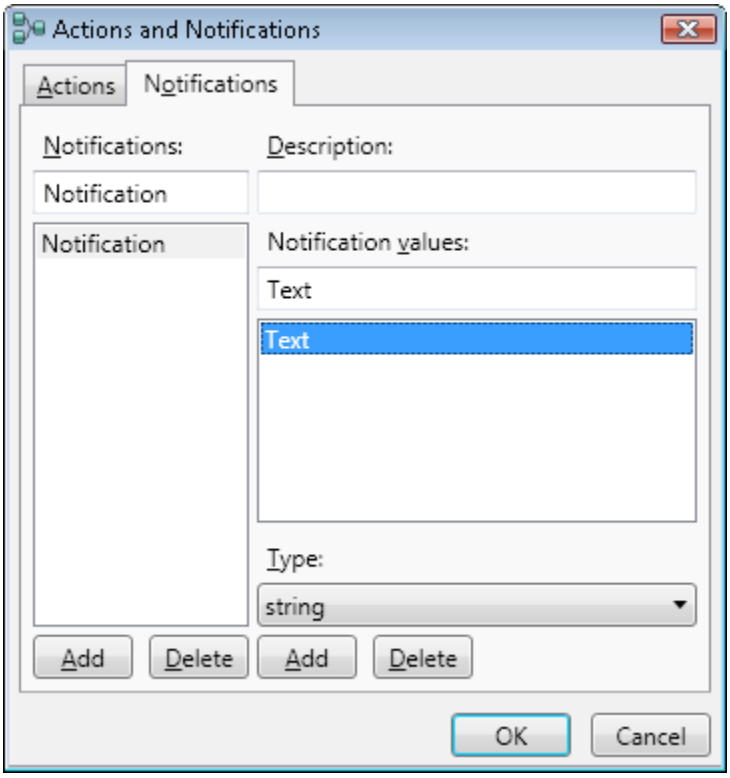

Figure 6 - Set up the Notification

You can now connect the output of the second **Merge** activity block to the round **Notification** connection pin on the right-hand border of the action page. Select the **Notification** you just created. Your activity's data-flow should look like the following.

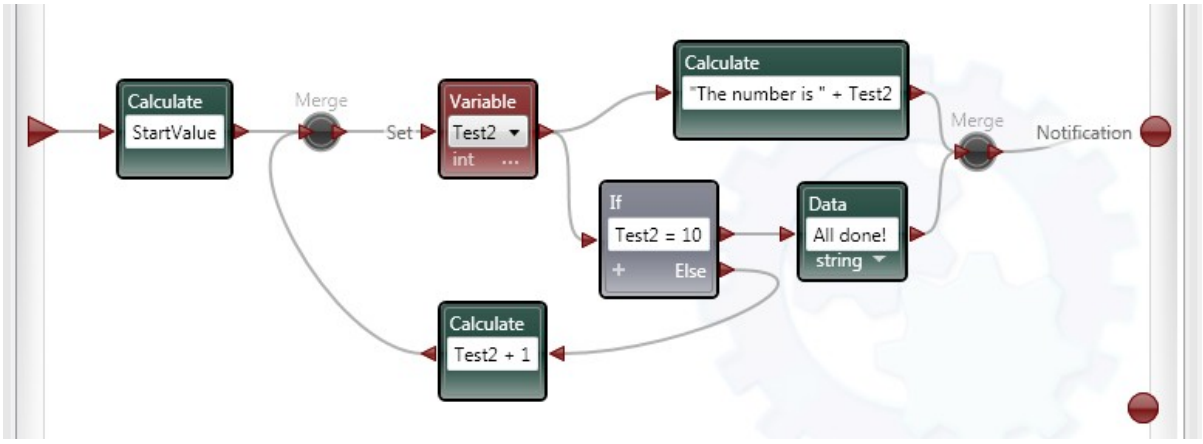

Figure 7 - Completed Action

Close the activity page or click the **Diagram** tab to switch back to the main diagram page. Connect the output of the **Data** activity block to your new **CountTo10** activity block.

Copy and paste the **CountTo10** block to create a duplicate of the activity you just created.

Finally, insert a **Text-to-Speech** service activity block and connect the round notification output of your **CountTo10** activity block to it. Set the connections as From: **Notification** (the name of your event) and To: **SayText**. Set the data connection from **Text** to **SpeechText**.

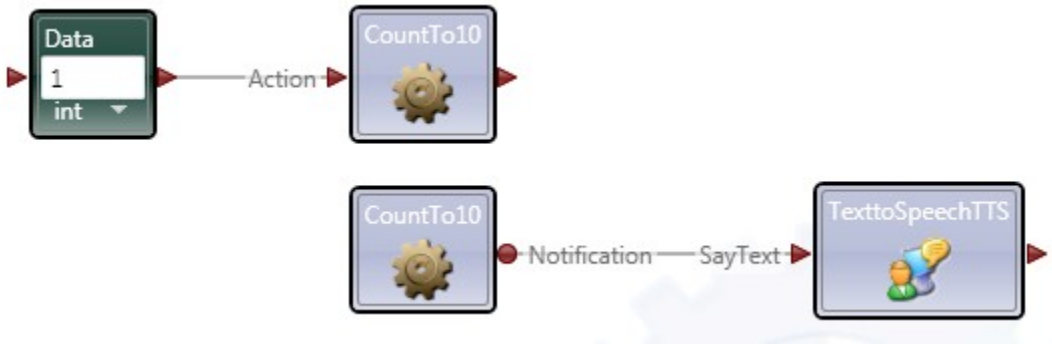

Figure 8 - Main Diagram

Run your application using the **Run** menu command (or by pressing **F5**). Your application should produce the same results as VPL Tutorial 2 - Incrementing a Value.

When you run your program, you might find that it does not count properly. Sometimes the numbers will be out of order and it will tend to talk over itsef. The first problem arises because messages can travel around the loop at different speeds and sometimes they catch up with each other.

The second problem occurs because new messages are sent to the Text-To-Speech service before it has finished saying the previous message. You could use the **SayTextSynchronous** operation instead of **SayText** because the synchronous operation does not send back a response until the speech is over. However, this would require changes to the program to wait for the Text-To-Speech service to finish after each message.

Counting without Loops and Variables

This tutorial shows how to create and use loops as well as how to use variables. In many situations you actually do not need variables. Loops are required in only a very few cases. The following example shows a much simpler way of counting without loops and variables. The activity called **RecursiveCountTo10**sends messages to itself. This process where an activity calls itself is called *recursion*. The key to making it work is that each time the action sends a message to itself, the message must contain a new value. It must also have a termination condition (in the **If** block) or it will run forever. The details of the example are left as an exercise to the reader.

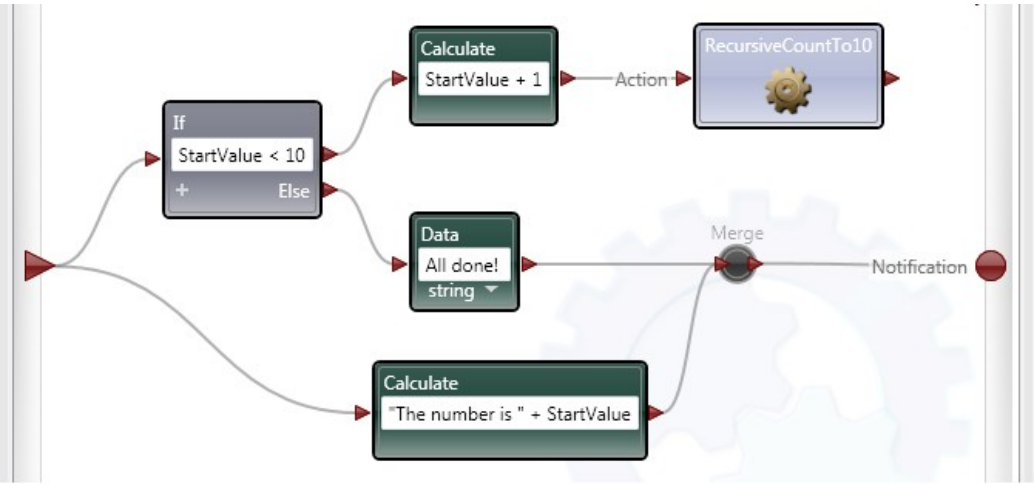

Figure 9 - Recursive Count

VPL Tutorial 4 - Run Simulation From VPL

This tutorial illustrates how you can start a simulation from VPL. To demonstrate this, you will create a VPL project that drives a robot around in a simulated world.

This tutorial is provided in the Microsoft Visual Programming Language (VPL) language. You can find the project files for this tutorial at the following location under the installation folder:

Create a Diagram to Run a Simulation

Drag a **Direction Dialog** from the Services toolbox into your diagram. The Direction Dialog provides a simple window with five buttons to control the robot: a left button, right button, Forwards button, backward button, and a stop button.

The **Direction Dialog** is designed to let you press a button on the dialog to trigger an event which passes the name of the button as a string. By connecting to this event, you can use an **If** activity block to control the motors of the robot.

Step 1: Define the Operations

Add a new **Switch** activity and a new **Calculate** to your diagram. The Switch compares its input message to the the values in its entries and sends the message to output of the first entry (from top to bottom) that matches the message.

Connect the **Calculate** block to the **ButtonPressed** notification of the **DirectionDialog** block and type **Name** in the text box to get the name of the button that was pressed.

In the **Switch** block, enter **"Stop"** (include the quotes). Click the Plus (+) button in the lower left of the activity block to display another condition. Enter, **"Forwards"**. Click the **Plus** button three more times and enter **"Backwards"**, **"Left"**, and **"Right"** in the text boxes.

Insert seven **Data** activities. Connect:

- One to the **"Stop"** output and entering **0.0**.
- One to the **"Forwards"** output and enter **0.8**.
- One to the **"Backwards"** and enter **-0.8**.
- Two to the **"Left"** output, entering **-0.6** and **0.6**.
- Two to the **"Right"** output, entering **0.6** and **-0.6**

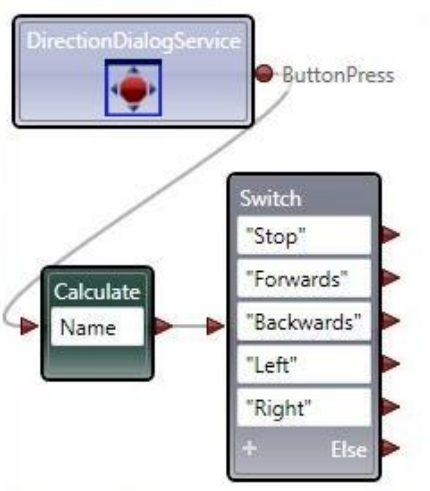

Figure 1 - Add the actions in the switch block

Insert two **Join** activities. Label the local variables in the Join text boxes as **Left** and **Right**. For each **Join**, connect the two **Left** Data blocks to one of **Join** activities and connect to the two **Right** Data blocks to the other.

The Join activities take the two data value inputs and send them together as a composite message.

Now connect the output of the first three Data blocks to a **Merge** block. Do likewise with the two **Join** blocks. Connect each of these new **Merge** blocks to a **Generic Differential Drive** service. In the first case, you will have to set both the Left Wheel and Right Wheel powers to **value**, i.e. they are both the same. For the second case, the Left and Right values from the Join will automatically be forwarded to the corresponding wheel powers. (This only works because the order in the Join matches the order in the drive request, not because VPL is smart about matching the fields up).

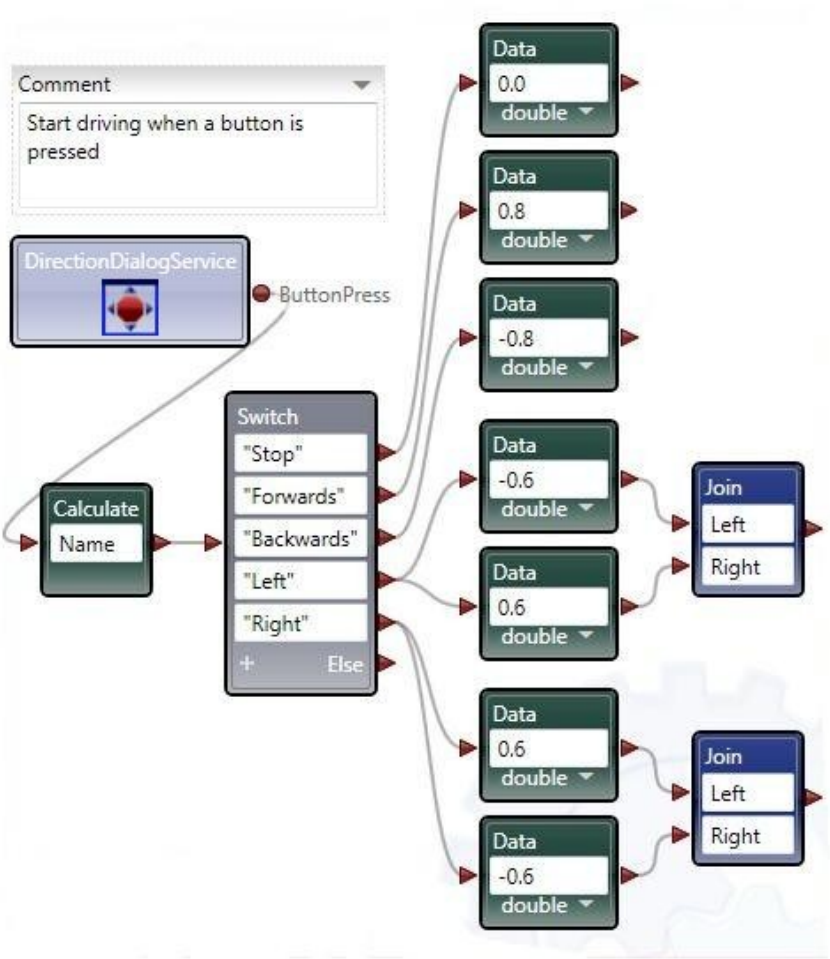

Figure 2 - For each action specify the motor power values

Your diagram is now complete and should look like the following.

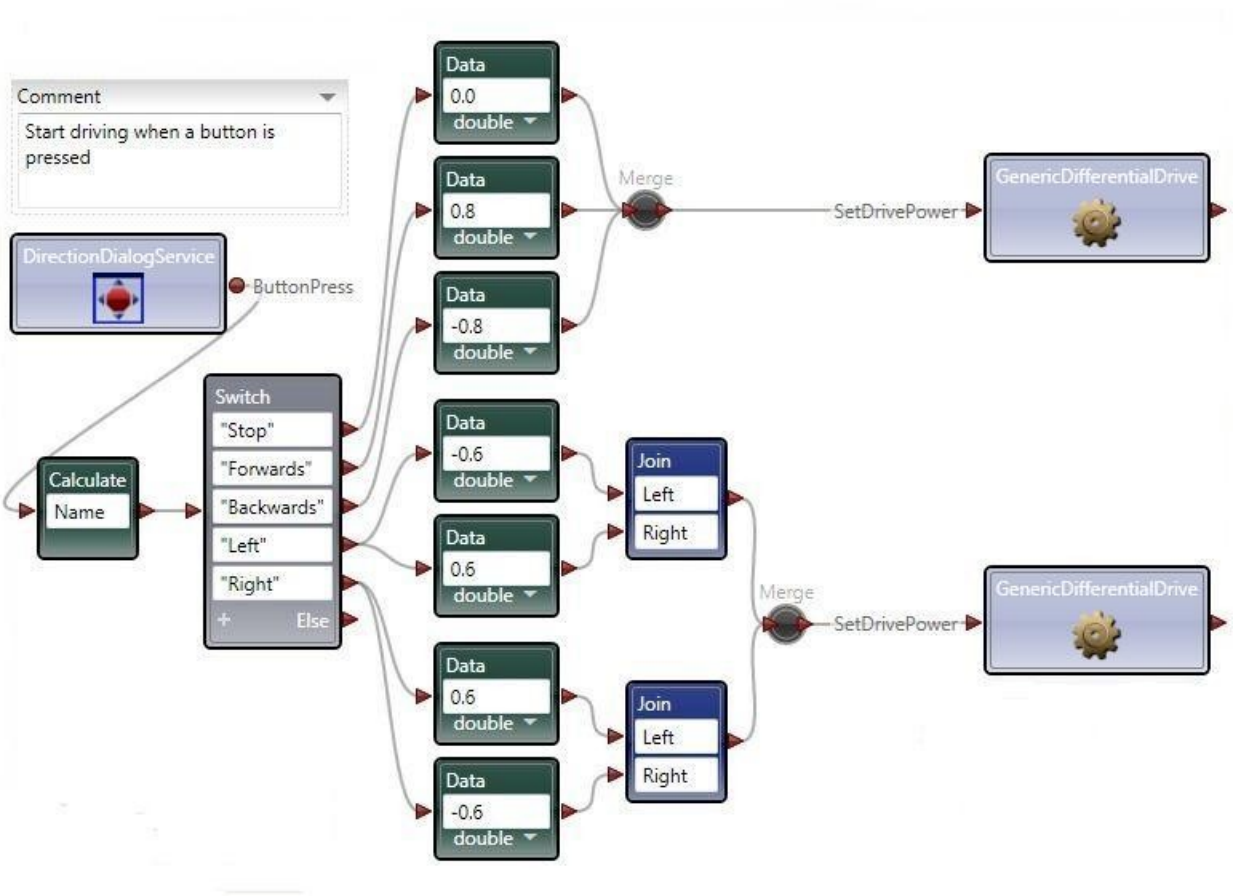

Figure 3 - Merge the values to set drive power

Step 2: Use Alternative Dataflow Diagrams

As a variation of this diagram, instead of using the **Stop** button, you can use the **ButtonReleased** event of the **DirectionDialog** service so that the robot goes when you press the buttons and stops when you release them. If you do not add this code, the robot will keep driving until you press the **Stop** button.

Figure 4 - Stop driving when a button is released

Select a Manifest

To connect to the simulated robot all you have to do is select the correct manifest for the **Generic Differential Drive**. If you have several **Generic Differential Drive** boxes on your diagram, you only have to set the manifest for one of them. Right click on the box and choose **Set Configuration**. From the drop-down list, click **Use a manifest**. Click **Import Manifest...** and choose: **LEGO.NXT.Tribot.Simulation.manifest.xml**, or **MobileRobots.P3DX.Simulation.manifest.xml**, or **SimulationTutorial3.manifest.xml**.

Run the Diagram

Now click **Run** on the **Run** menu (or press **F5**). If you have not saved your project yet, VPL opens the Save dialog box. Type a name for your project and click **Save**.

VPL now proceeds to run your application. If you get a message asking whether to unblock the application, click **Unblock**. You should see the simulation window (and the DirectionDialog box if you used it). Now, depending on which diagram you built, you can either use the triggers from your Xbox controller to drive the robot or the buttons on the dialog box.

To stop your application, click the **Stop** button in the Run dialog displayed by VPL.