Creating Blocks for LEGO Mindstorms EV3 08202013_01

Introduction

This document is intended to help you build your first simple sensor or data operations blocks for use in LEGO Mindstorms EV3. Blocks are (pun intended) the building blocks of all EV3 programs. They live in the palette across the bottom of the EV3 window and are separated into families such as **Action** and **Sensor**. Each block you find in the palette can and usually does represent more than one operation that is referred to as a **Mode**. In block developer terms, The "**Block**" you find in a Palette Category is actually a **PolyGroup** that contains one or more **Block** items, each representing a particular **Mode**. Check out the section of this document about **Blocks.xml** for a more thorough breakdown. Additionally, the package you create for your block or module (collection of blocks) is a collection of files and folders with specific responsibilities and expectations.
EV3 is based upon WebUI Builder, a product from NI used to create viewers for web-based data sources. It has some similarities to LabVIEW, like palettes, wiring, block diagrams, etc. so when we refer to some feature and say "WebUI Builder" it is safe to substitute "LabVIEW" in your mind to help with understanding. One difference, though, is that programs use a **VIX** file extension instead of **VI**. We may use the term **VI** in this document and even in the dev environment menus but, technically, we are always referring to **VIX** files. **VI**s don't exist in EV3 and, unfortunately, there's no tool to convert **VI**s to **VIX**s.

A. **What is a block?**

1. **Anatomy of a Block**

a. **The top directory**

This directory gets its name from the **ModuleName** attribute either of the blocks.xml files, one of which resides in this directory. Since there are two files named blocks.xml, we will refer to this one as the "main" blocks.xml. The blocks.xml files are discussed in depth below.

b. **The "VIs" directory**

This is where you put all of the code for your block. The general guidance is to put hardware specific code in either of the subdirectories, PBR or NXT, depending on which brick it supports. *Note:* ***PBR*** *is an abbreviation of* ***PBrick****, which is the internal name for the brick we now know as the EV3. In this document and other discussions,* ***PBrick*** *means* ***EV3*** *and vice-versa.* Code that not hardware dependent belongs in the parent "VIs" directory. For example, VIs that actually talk to the hardware or support talking to the hardware go in the brick-specific directories while code that scales the data or prepares it for display might go in the main VIs directory. The VIs available at edit time change depending on what type of brick is connected. When you are connected to a PBrick your VI.lib consists of WebUI Builder /LabVIEW's VI.lib plus the VIs is your "VIs" directory plus the VIs in your "PBR" directory. If you connect to an NXT brick, the PBR VIs are unloaded and those in the NXT directory are now available. This is important to understand (especially for sensor blocks) since the top-level VIs for each of your block modes will most likely be different for each target brick type. For each mode, 1 and only 1 VI can be named in the blocks.xml file as the VI for that mode. We dynamically switch out the PBR and NXT directories to cut down on the complexity of the blocks.xml file and to help with memory consumption.

i. **"PBR" subdirectory**

Place code that is PBrick/EV3 specific in this directory.

ii. **"NXT" subdirectory**

Place code that is NXT specific in this directory.

iii. **What if my sensor only supports one brick type?**

In this case you must tag the affected modes as target specific in the main blocks.xml and place empty do-nothing VIs of the same name in the unsupported targets directory. The easiest way to create these is to copy the supported target VI to the unsupported targets directory and then delete its diagram code, leaving it inputs and outputs and connector pane identical to the other. If it does not match up you will get errors.
You may be tempted to just put everything in the VIs directory and ignore the rest, for example if you never plan on supporting the NXT. This will probably work for the intended target but the behavior on the unsupported brick will be undefined and could cause problems.

c. **The "help" directory**

There are per-language subdirectories named using the language-country code designation (such as "en-US"). Supported languages are listed in a table later in the **Blocks.XML Notes** section.
The main help file for your block derives its name from the **ModuleName** just as the main directory for the block does. An html file in the form <*ModuleName*>.html is expected and required for a proper fit into EV3.
Note: Though it may seem convenient to place images used in your help file in the block "images" folder, don't! Place all content related to your help in this help directory. You may create subfolders if you like, however, but steer clear of using content outside of the help directory. There is some magical merging and mapping that takes place internally to make your block fit in to the environment and this makes linking to things outside of the help directory difficult and non-intuitive.

d. **The "images" directory**

This is where the images for your block live. These are used in the palettes and on the block's configuration. All of these images have required names and sizes.

e. **The "strings" directory**

There are per-language subdirectories named using the language-country code designation (such as "en-US").
Each language folder contains a "blocks.xml" file that is used to provide user visible names, context help, and help URLs for the programmatic items defined in the main blocks.xml file. For clarity, we will refer to this as the "strings" blocks.xml file.

2. **What does a block look like?**
There are many images involved in displaying a block, its terminals, and other features. Blocks and/or their modes show up in the palettes, on the programming canvas, in the hardware page, and possibly other places. We’ll restrict this discussion to the very basics

and advise motivated block developers to inspect other blocks for examples not covered here. Note: image names are inferred from other named entities within the blocks.xml file and cannot be overridden. Also, images sizes are very important. Use the sizes prescribed or the images may be cropped, resized, or distorted.

a. **Palette image requirements**
   i. This is the image that is overlaid on the main palette block icon for a given category.
      A. Size – 20X20
      B. Name – **PolyGroup_**<*PolyGroup-Name*>**_Palette.png**
      C. There is a second version of this image with "**_MouseOver**" appended. This is to help indicate that the mouse is indeed hovering over the icon. Take a look at the mouse-over images for other blocks/images for ideas about how to handle yours. To get started, you can just use a copy of the original non-mouse-over image.

b. Diagram image requirements
   i. This is the left-most image on the block after being dropped from the palette. It is similar to the palette version except for size.
      A. Size – 34X34
      B. Name – **PolyGroup_**<*PolyGroup-Name*>**_Diagram.png**

c. **Mode configuration image requirements**
   i. These are the images used for the mode and mode selection menu.
      A. Size – 38X22
      B. Name – **PolyGroup_**<*PolyGroup-Name*>**_Mode_**<*Mode-Name*>**_Diagram.png**
      C. Supports mouse-over as in the Palette image above.

d. **Hardware Page image requirements**
   i. These are a set of images used to build the graphical menu in the **Hardware Page**'s **Port View** tab. When you click on a port in this view, a menu of possible devices appears, either sensors or motors depending on the port you click. The image next to the sensor name is the **Category** image. The submenu that appears when you hover over the **Category** is created using the **Mode-Hardware** images. Finally, the images displayed in the Port after you have selected a sensor (or after Autodetection has discovered a sensor) are a combination of the **Diagram** image and the **Mode-Hardware** image.
      A. **Category** image
         A. Size – 38X22
         B. Name – **PolyGroup_**<*PolyGroup-Name*>**_Category.png**
         C. Supports mouse-over as in the Palette image above.
      B. **Mode-Hardware** image
         A. Size – 22X22
         B. Name – **PolyGroup_**<*PolyGroup-Name*>**_Mode_**<*Mode-Name*>**_Hardware.png**
         C. Supports mouse-over as in the Palette image above.

3. **Blocks.xml, the files that tie it all together.**

   This topic is big and important enough to warrant its own separate section. See **Blocks.XML Notes.**

B. **Tutorial - Building a simple sensor block**

Read the **Using EV3 for Block Development** and **Blocks.xml Notes** sections prior to starting this tutorial. We will use the **SampleSensor** block that came with this package and discuss the steps you would take to build your sensor block from it. You should import the SampleSensor.ev3b block into your dev environment. This way, you can refer to the block in the environment while inspecting its source files and reading the documentation.

1. First some planning – choose some names to get started. Refer to the **Blocks.xml Notes** section for more info on these terms. You will need…
   a. **ModuleName**
   b. **PolyGroup** Name
   c. **BlockFamily** (probably **Sensor**)
   d. Names for your **Block Modes** and corresponding VIXs – we recommend starting with the very basics at first, as we have done in the sample. We have three initial modes and VIXs to satisfy the basic requirements for a sensor block.
2. Decide where you are going to store your source files and copy the **SampleSensor** folder into that location.
3. Rename the folder using the **ModuleName** you just chose.
4. Update your main blocks.xml with the info you decided upon earlier.
5. Now, you may as well update your strings blocks.xml file to reinforce the good habit of keeping them up-to-date and in sync with each other.
6. Change the image file names to match your PolyGroup, Mode, etc. names.
7. Remember that you will eventually need to update and complete the help file for your block. It will probably be easier to do this later, once your block is more complete, rather than spending a lot of time and energy keeping it up-to-date.
8. Now let's put some code in place. There are quicker ways to accomplish what we're about to do but these steps will give you some practice using the environment.
   a. In a new project and program, connect to an EV3 brick.
   b. Drop the SampleSensor block and ensure that the Measure Value mode is selected.
   c. Double-click the block and the implementation VI, SampleSensorValue.vix, will open.
   d. Select Export from the Developer menu and save this file into your block's VIs\PBR folder. Name it using the name you decided on earlier. Remember, this file is pointed at by the blocks.xml file so the names must match.
   e. On the diagram, find the subVI named SampleSensorEV3Core.vix and double-click it.
   f. Export this to your PBR folder using any name you like. Using a pattern similar to the SampleSensor, which includes the target brick type and the word "Core" will help others understand the purpose of this subVI better. This "Core" subVI technique 1) acts as a mutex around the code that actually deals with the hardware and 2) allows this underlying HW code to change without messing up the top-level block too much. The mutex is the most important concept. Parallel loops trying to read from the same sensor type can be very problematic without some sort of mutex.
   g. Now restart the dev environment without saving anything. You have been exporting to your block's source directory so you haven't changed anything about the SampleSensor but now is the time to get it all out of memory just to be sure.
   h. Once it has restarted, create a new project, and connect to an EV3.
   i. Select Import from the Developer menu to open your measure value VIX.
   j. Do the same for your "Core" VIX.
   k. This next step demonstrates using the project pane to drop something that doesn't live in a palette.

        i. Our goal is to replace the SampleSensorEV3Core.vix with your "Core" VIX.
        ii. Since your "Core" VIX is open, it has been added to the internal project that you can access using the Project pane. If it is not visible, see the section **Using EV3 for Block Development** for instructions on how to show the various panes you might need.
        iii. In the project pane, find your "Core" VIX and drag it onto the diagram of your measure value VIX.
        iv. Now just delete the old Core VIX and move yours into place and reconnect the wires.
        v. Export your measure value VIX to your **PBR** directory to save this change.
    l. Once again, let's shut down the environment to get a clean slate. Don't bother restarting for now.
9. At this point you should be in a position to try importing your block for the first time, so let's create the block import file. Our instructions use WinZip but any compression utility that creates standard zip files and stores relative directory information can be used.
    a. Right-click on the top-level folder for your block – the one you renamed using your **ModuleName** – and choose "Add to <yourfoldername>.zip".
    b. To quickly test this file, copy it to an empty folder somewhere, right-click it, and select "Extract to here". If it recreates your sensor folder and its subfolders and files, then it is zipped correctly.
    c. Replace the "zip" file extension with "ev3b", which stands for **EV3 B**lock.
10. Start up your dev environment, create a new project, connect to an EV3, and try importing your block using the Block Import Wizard. If successful you should be able to drop your block, select the measure value mode, and even run it though the "Core" VIX at this point just returns a random number.
11. Before going any further, you might consider editing the images for your block to help you differentiate between it and the SampleSensor block.
12. Once you have edited the images, recreate the "ev3b" file. Before importing your block again, use the Manage tab in the Block Import Wizard to remove your old sensor block first.
13. You can now continue to edit your Core VIX to actually talk to your sensor or you can continue to implement the remaining basic modes for your sensor and save the "Core" functionality for last.
14. Be sure to remove any files named with SampleSensor from your block directory as you generate versions of them for your block. Removing these files shouldn't cause any problems at all. If it does, then for sure, they're the kinds of problems you want to find out about now instead of later.
15. Finally (and again), don't forget to take care of the help file ☺

C. Tutorial - Building a simple data operations block

Building a simple data operations blocks is very similar to building a sensor block. Instead of reproducing the steps from above we'll point out some of the major differences you'll find when you apply the sensor block steps above to the SampleDataOp block from this package.

1. Deciding on the module, block, and mode names is still the first step. For a data operations block, though, there is not the expectation of standard measure, compare, and change modes. Our sample just provides two more math operations, Reciprocal and Cube.
2. In many cases, data operations are not brick specific so your VIX code can be placed in the VIs directory instead of the PBR and NXT directories.

3. Notice that there are also fewer images to create since data operations blocks do not show up in the hardware page.
4. The main blocks.xml file is a little simpler than that of a sensor block. Typically, you won’t find elements or attributes like **CompilerDirectives** and **ValueDisplay**.

D. **How do I communicate with sensors and perform things like File I/O on the brick?**

Unfortunately, there’s not a lot of plain and simple documentation for the EV3 firmware at the time of this writing. In the mean time we recommend inspecting the blocks that ship with EV3 using the supplied development environment and copying code snippets from them as necessary. Also, in the OtherFiles folder in this package, we have included two VIXs containing many of the low level function calls we used in building the blocks that ship with EV3. They are full of what we lovingly refer to as “gray blobs”. To use these, create a new separate project and open either or both of these files, find the gray blob you need, copy it to your clipboard, and then paste it into your program. Do not open these files in the same project you are working in as they will be treated as part of your project and will break your program since they are just code snippets, they’re not runnable.

Blocks.XML Notes

Introduction

Two "blocks.xml" files are used to define almost everything about a block or module except the code that gets compiled and used in a program.
The first lives in the uppermost or root directory for the block and defines such things as the subVIs to use for a given mode, the parameters for a given mode, and which mode to use as the default when you first drop the block from the palette.
The second lives in a subfolder of the "strings" directory. It maps some of the programmatic names in the first blocks.xml to display names and provides context help text. This file is duplicated, translated, and stored in language specific directories for localization purposes.
Refer to the blocks.xml files for the SampleSensor while reading this.

These files are referred to as the "Main" and "Strings" blocks.xml files, respectively.

A. *Main* blocks.xml

The following is a brief description of the key elements of the Main blocks.xml and their attributes. *Note: If you inspect the main blocks.xml for the LEGO module (that ships with EV3) you will find many more items than are described here. A good number of these are no longer used but were left in place to avoid issues where the parser expects to read something even though it is ultimately unused.*

- The topmost element is **EditorDefinitions**. It is just a container for the rest of the document.
- Next is the **PolyGroups** element. It has two key attributes, **ModuleName** and **ModuleVersion**.
  - **ModuleName** is the name that will show up in the Block Import/Export Wizard. It represents the unit that will be installed or uninstalled using this tool. Our samples demonstrate defining a single block within blocks.xml so that the user has the flexibility of adding or removing a single block at a time. You can define more than one block by adding more **PolyGroup** elements inside the **PolyGroups** element but doing so will prevent individually managing the blocks.
  - **ModuleVersion** is simply a number to display in the Block Import/Export Wizard. It is not used programmatically in the environment.
- The **PolyGroup** element defines a single block and its modes. It has two key attributes, **Name** and **BlockFamily**.
  - **Name** is the programmatic name for the block. It is used in forming certain file names that the environment expects. In our samples, this matches the **ModuleName** above but that is not a requirement.
  - **BlockFamily** determines the palette in which the block resides. Current valid families are **Action**, **DataOperations**, **FlowControl**, **Sensor**, and **Advanced**.
  - Within the **PolyGroup** element are a couple of elements common to all blocks, **Parameter** and **Block**, and a third, **Hardware**, which only applies to sensor blocks.
    - First, all the **Parameter** elements are listed. The various attributes of a parameter specify its datatype, default value, direction, and more. This list is a superset of all the parameters for all the modes of a particular block. The subset of parameters for a given mode is specified in the **Block** element later.
      - **Name** – This is the actual name of the terminal in the mode VIs. The user-visible name is specified in the other “strings” blocks.xml.

- **Direction** – **Input** or **Output**
- **DataType** – This should be **Single** for all public parameters, those that can be wired to or from the block on an EV3 diagram. Special parameters that are used internally can be other datatypes. These special parameters include the Direction parameter used in wait for change blocks, the Comparison type used in the wait for blocks, and the "i" parameter used internally to plug into loops, and possibly more. In these cases you should just use the types we have provided in the samples as they are correct and expected by the EV3 environment.
- **DefaultValue** – This is simply the default value for the parameter when it is dropped from the palette. This attribute is used when the default is the same for Education and Retail versions of EV3. You should either use this attribute or the edu/retail pair of attributes, not all three.
- **EducationDefaultValue** – Default value in EV3 EDU. See **DefaultValue** above.
- **RetailDefaultValue** – Default value in EV3 Retail. See **DefaultValue** above.
- **MinValue** – This is the minimum value you can type into this parameter on the block's configuration. Out-of-range values will be coerced when you enter them. It does not affect values that are wired into this parameter.
- **MaxValue** – This is the maximum value you can type into this parameter on the block's configuration. Out-of-range values will be coerced when you enter them. It does not affect values that are wired into this parameter.
- **Configuration** – This attribute specifies the behavior or control you get when you click on a parameter to change it on the block's configuration. For example, the slider that you use to change motor power is called "SliderVertical.custom". Another possibility is to point at an XML file that defines values, names, and an image suffix that is used to determine the picture for each ENUM value. Take a look at the "Identification_ComparisonType.xml" file that can be found in <your_install_directory>\Resources\Blocks\LEGO\images. Any of the built-in custom controls or XML defined sets of images can be used in a third-party block by using the "builtin://" prefix before the filename as we have done in the blocks.xml for our samples. You can also create your own XML-defined image sets, just use one of ours as an example and place your files in your "images" directory. This attribute is optional, there are default behaviors for numbers, strings, and Booleans.
- **Identification** – This is similar to **Configuration** except it affects the image displayed above the wiring terminals on the block. It can point to something as simple as a PNG image or it can also use an XML defined "recipe" to determine what to display. A good example is the Steering parameter on the Move block. As you move the custom slider the Identification image changes according to the value of the slider per the definition in the "Identification_Steering.xml" file. You must, at least, specify an image. Otherwise, a question mark will appear in the **Identification** area.

        - **CompilerDirectives** – This is a misnomer. It's really more of a marker for special terminals. For example, comparison modes require that a Boolean "result" be returned. Instead of insisting that this parameter be named Result and locating it by name we chose to mark it with the **CompilerDirectives** attribute "Result". This allows you to name these parameters as you prefer and simply "tag" them with a **CompilerDirective**.
        - **ValueDisplay** – This is not used that often. It uses the same XML definitions as **Configuration** and **Identification**. A good example of this in use is to drop a Switch block and choose the "BrickButtons>Measure>BrickButtons" mode. The images that appear in the top of each case are driven by the **ValueDisplay** attribute.
    - Next, if this is a sensor block, we have the **Hardware** element. The three elements it contains that are important for third-party block developers are **EV3AutoID**, **Direction**, and **DefaultPort**.
        - **EV3AutoID** – Unless your sensor is a new EV3 sensor that supports EV3 automatic identification, this should be -1, otherwise it should match your sensor's AutoID number.
        - **OtherAutoID** – Use this if you have more than one AutoID-able sensor and you want them to map to the same block.
        - **Direction** – **Input** or **Output** depending on the hardware.
        - **DefaultPort** – This specifies the default configuration for the port when dropped fresh from the palette and the sensor is not plugged in. If the sensor is plugged in or if the hardware page is configured for a particular sensor, then the block configuration will attempt to match the hardware page.
    - Next, we define all the block modes using **Block** elements. Each **Block** element represents a selection available in the block's mode chooser.
        - **Mode** – This is the programmatic name for the mode. It is also part of the recipe for some of the help information. Check out the help file for the SampleSensor for further explanation of the help system's requirements.
        - **Reference** – The attributes for this element point to the resource that implements the code for a given mode.
            - **Type** – Always **VILib**. There are other possible types but they're not important for third-party blocks.
            - **Name** – This is the name of the VI for this mode.
        - **ParameterReference** – These list the actual parameters used for this mode using the Name attribute.
            - **Name** – This string points to one of the **Parameter** elements at the beginning of the **PolyGroup**.
        - **PaletteInfo** – There should only be one **PaletteInfo** element in a **PolyGroup**. The **Block** that contains this element becomes the default mode for the block when it is dropped. Its only attribute, **Weight**, determines where it appears in the palette from left to right.
        - **BlockInterface** – **Measure**, **Compare**, or **Change** – By setting this element you are stating that the block is ready to plug into EV3 in the mode specified. Our Sample Sensor block has everything you need to meet the **Measure**, **Compare**, and **Change** interface requirements. Remember those **CompilerDirectives** mentioned above? They indicate the special parameters used to meet the interface requirements.

- **Flags** – Though you may find others about, the only flags you are likely to use are **PBROnly** and **NXTOnly**. Flags don't have values, they are either present or not.
    - **PBROnly** – Use this flag to indicate that the mode only works on EV3 bricks.
    - **NXTOnly** – Use this flag to indicate that the mode only works on NXT bricks.
- **HardwareModeInfo** – Used for HW Mode detection and setting as well as provides info for datalogging if applicable.
    - **Name** – Used in datalogging as part of dataset name.
    - **ID** – Refers to the mode of the sensor.
    - **Range** – Used in datalogging.
    - **Unit** – Used in datalogging.
-

B. *Strings* blocks.xml

The "strings" blocks.xml doesn't require a great deal of explanation. You'll notice the same or very similar element structure with the addition of a few new attributes and elements.

- The **PolyGroup** element has two new attributes, **DisplayName** and **DisplayNamePrefix**.
    - **DisplayName** – This is the user visible name for this block/polygroup. The **Name** attribute is the link to the main blocks.xml entry.
    - **DisplayNamePrefix** – This string is used in Education Datalogging for such things as dataset names, axis labels, etc. It is concatenated with other info to form unique identifiers where necessary.
- The **Parameter** element has a **DisplayName** attribute as well, and a **Description** element.
    - **DisplayName** – Same as above.
    - **Description** – This string is displayed in the Context Help window when you hover over this parameter with the mouse.
- The **Block** element has **DisplayName** and **Description** items as above. Their functions are nearly the same.
- Beyond these special items, and more generally, the strings blocks.xml file resides in a language/locale specific folder. If you choose to localize your block, simply duplicate the blocks.xml in the **en-US** folder, translate the special string items discussed above, and store them in an appropriate folder within the **strings** directory.

Language	**Code/DirName**	**Education**	**Retail**
Chinese	zh-Hans	✓	✓
Danish	da	✓	✓
Dutch	nl	✓	✓
English - UK	en-GB	✓	✗
English - US	en-US	✓	✓
French	fr	✓	✓
German	de	✓	✓
Italian	it	✓	✗
Japanese	ja	✓	✓
Korean	ko	✓	✓
Norwegian	nb-NO	✓	✗
Portuguese	pt	✓	✗
Russian	ru	✓	✓
Spanish	es	✓	✓

Swedish	sv	✓	✗

Using LEGO Mindstorms EV3 for Block Development

Introduction

We've created a special version of EV3 that enables extra menus and functionality needed to edit VIs at a deeper level than the normal robot programs. It basically exposes some of the functionality that you would find in our WebUI Builder but it's much less polished.

A. **Installing the Dev Version**

For now, just unzip the EV3BlockDev.zip file inside the DevEnvironment\Windows directory and run the MindstormsEV3.exe. You must have already installed EV3 so the drivers are present. Eventually we'll package it better.

B. **Environment Basics**

Now, show the ribbon if it is not already visible. Click the **New Programming Project** button and wait for a new project to be created, then click the **Show Ribbon** item in the **Developer** menu. Note that most menu items are disabled and gray unless a project is open. This is not a WebUI Builder "feature" rather it is a behavior of EV3 that we can't turn off, even in dev mode.

- Showing the palettes and other useful items – In the ribbon, go to the **View** tab and click the **View Panes** button. Three items I this list will be most useful for block development.
  - Palette – This turns on the WebUI Builder palette. Its default placement is to the left of the EV3 programming canvas. If you've built blocks for NXT or otherwise used LabVIEW before, the items in the palette should look familiar. **If you've done neither of these we suggest seeking some LabVIEW / WebUI Builder basic training before moving on.**
  - Project – Everything you work on lives in a project. This pane gives you access to whatever may be loaded for your project at any time. It is useful if you need to drop something on a diagram that doesn't live in a palette already. You would open the VIX you want to drop then go find it in the project pane and drag it from there on to your target diagram.
  - Errors and Warnings – Eventually you'll want to compile and run your code, that is, test your block. If the download fails, this pane should contain some information for you to use to debug and fix the problem.
  - The rest of the items available in the View Panes list may or may not work or offer any value to the block development task. Some are leftovers from early development and testing.
  - Note that the panes discussed above can be pinned in place or unpinned and allowed to collapse themselves until activated.
- The Importance of the **Developer** menu - Almost everything you will need to do concerning building your block is handled within the Developer menu (and sometimes the ribbon). The two most important items are New VI and Export.
  - **New VI** – This generates a new blank VIX.
  - **Export** – This is the function you use to save your work. The **File>Save** and related items have all been over-ridden by EV3 and can only be used to save EV3 programs and projects. So, instead of invoking **File>Save** or pressing **CTL-S** to save your work, you will be going to the **Developer** menu and selecting Export (a lot).

    - Other items in this menu are redundant to other menus or ribbon items or don't apply to block building. Some may cause problems so **Export** often and don't explore while trying to get something important done ☺.
- Other Development features
    - If you double-click on an EV3 block in an EV3 program, the implementation VIX for the block's selected mode will open. This is useful not only for working on your block but to peek inside other blocks to see how they are implemented.
    - You can hold down the CTL key and use the mouse wheel to zoom the diagram in and out.
    -