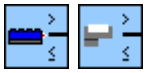
Intermediate & Advanced ROBOLAB

Table of Contents

FORKS

Some of the icons…


Touch Sensor Fork / NXT Touch Sensor Fork


Light Sensor Fork / NXT Light Sensor Fork


Fork Merge

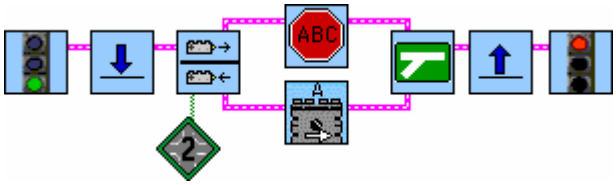
Description…

A Fork is equivalent to an "if-then" statement. If a certain condition is met (i.e., if the touch sensor is pressed in), then the program does one thing; if not, it does something else. There are various types of forks, such as a touch sensor fork, light sensor fork, rotation sensor fork, container fork, etc.

All forks need a corresponding Fork Merge icon at the end to connect the two paths back together. If you have multiple forks, each fork must have a corresponding Fork Merge icon.

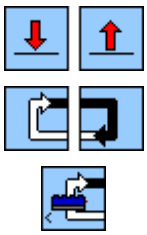
Example…

In this example, a touch sensor fork is used. If the button is pressed, then motor A runs forward. If the button is not pressed, then all motors stop. This is repeated over and over again.



LOOPS

Some of the icons…



Land /Jump

Start of Loop / End of Loop

Loop While Light Sensor Is Less Than

Description…

There are two major types of loop structures in Robolab: Jump/Land and Start/End of Loop.

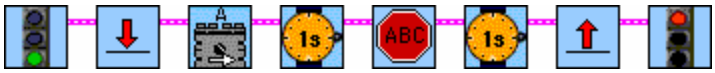
When a program encounters a Jump icon, it skips all other codes and jumps right to the Land icon with the same color. Thus, if a series of commands are placed between a Land and a Jump, that segment will be repeated forever. Jump/Land are similar to Goto functions in C or BASIC.

The Start of Loop and End of Loop is used to repeat something a specific number of times. The Start and End icons are always used in pair, and any commands placed in between the two icons are repeated for a specified number of times. A modifier is wired into the Start of Loop icon to specify the number of repetition.

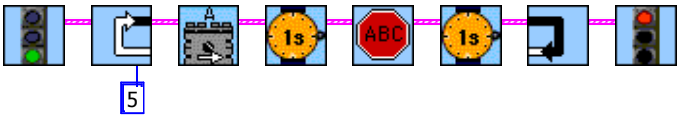
There are other types of While loops available, such as the Loop While Light Sensor Is Less Than icon which will repeat the commands until the light sensor value goes above a specified number. All Loop While icons are used in conjunction with an End of Loop icon.

Example…

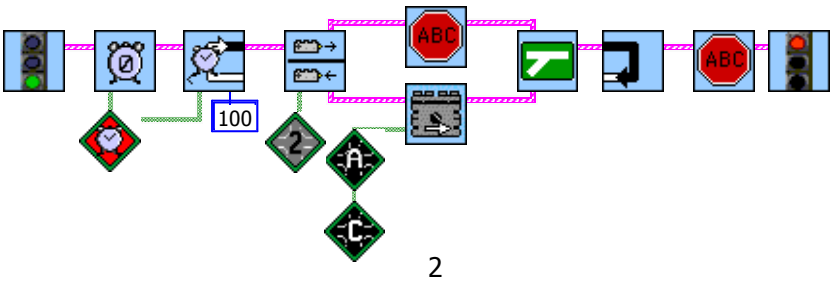
In this example, motor A is turned on for one second, then stopped for one second. The program then jumps to the beginning, then repeats over and over endlessly.



This example is similar to the previous program; however, it only repeats the code five times as opposed to repeating forever.



The following example uses a "Loop While Value of Timer is Less Than" icon. This program lets the user move a car by pushing a touch sensor and stop it by releasing the sensor, but only for 10 seconds (100 × 0.1 seconds). After 10 seconds, the program stops all motors. Timers need to be zeroed at the beginning of the program (or during a program, whenever you want to start counting).



TASKS

Some of the icons…


Task Split


Start Task


End Task


Task Priority

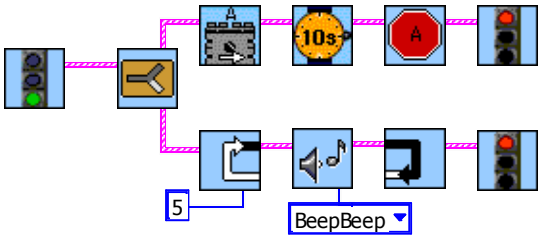
Description…

Tasks are used when you want the program to do multiple things at the same time – for example, using light sensors to follow a line while playing music. Task Split is used to create a new task. Tasks are executed simultaneously unless you use Start Task, End Task, or Task Priority commands to control them.

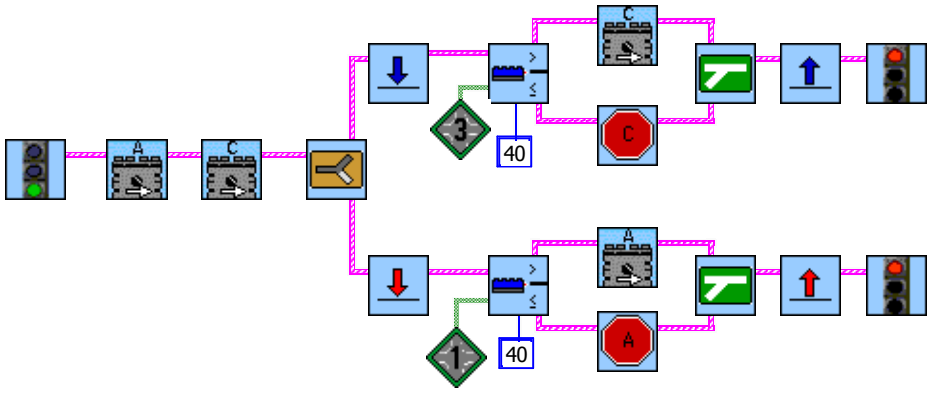
Unlike Forks, Tasks do not merge back together. Instead, each task gets its own End icon (the red light).

Example…

The first example uses task split to play a sound and move a motor simultaneously. The program runs motor A for 10 seconds, while it plays the "BeepBeep" sound five times.

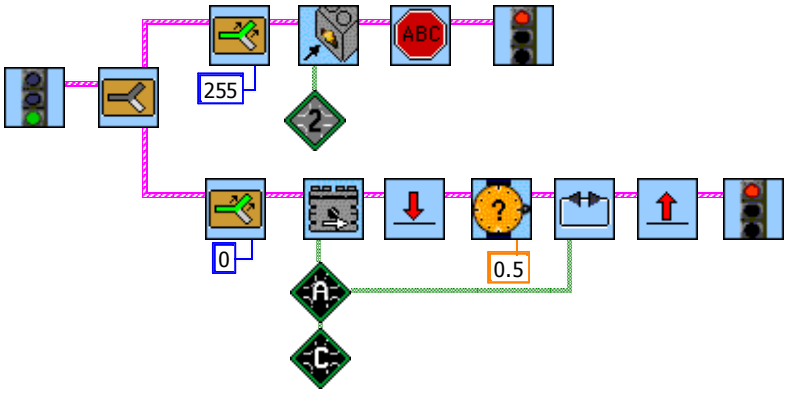


The example below is a two-sensor line follower program, where sensors on right and left sides of the car keeps the car straddled over a black line. In this program, task split is used to monitor two light sensors simultaneously. Each light sensor controls one motor. Notice that each task ends with its own red light.



The third example uses Task Priority to control which of the tasks are executed. Task Priority icon assigns a priority number to a task (number can be anywhere between 0~255; highest number has highest priority). Tasks with higher priority overrides lower priority tasks except when the higher priority is waiting for something.

In this example, the high-priority task (255) is waiting for a touch sensor. During that time, the low priority task is executed (motors turn on and flips directions every 0.5 seconds). As soon as the touch sensor is pressed, the high priority task overrides the low priority task and stops all motors.



EVENTS

Some of the icons…


Set Up Pressed Event / NXT Set Up Pressed Event


Start Monitoring for an Event


Stop Event Monitoring


Event Landing


Red Event

Description…

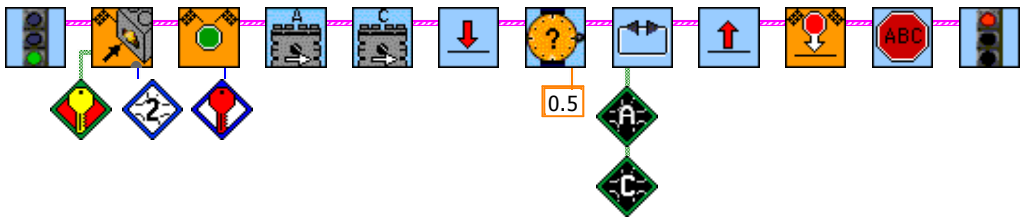
Events are used as triggers to make the program jump to a different place. An Event always needs to be defined using one of the Set Up Events icons. An "event" could be many different things – a pressed button, light sensor that goes over some value, a rotation sensor registering a certain value, etc.

At any point after the event has been set up, you can Start Monitoring for an Event. When the event is triggered, the program jumps to the Event Landing icon.

You can monitor for multiple events at the same time. When you have more than one event, each event setup, monitoring, and landing are defined by modifiers (i.e., Red Event) to differentiate them.

Example…

The example below uses a touch sensor event. At the beginning of the program, a Pressed Event is configured with a Red Event key and Port 2. After starting to monitor for the red event, the motors are turned on, and reverse directions every half second. The car will keep on going back and forth forever until a touch sensor is pressed. When the touch sensor is pressed, the event is triggered, and the program jumps out of the loop to the Event Landing icon and stops all motors.



Note, this program does exactly the same thing as the third example in Tasks. As with most engineering solutions, there are more than one ways to do the same thing.

Events are useful for creating overriding programs such as following a line until you bump into a wall.

CONTAINERS

Some of the icons…


Fill Container


Red Container


Value of Red Container


Light Container / NXT Light Container


Add to Container

Description…

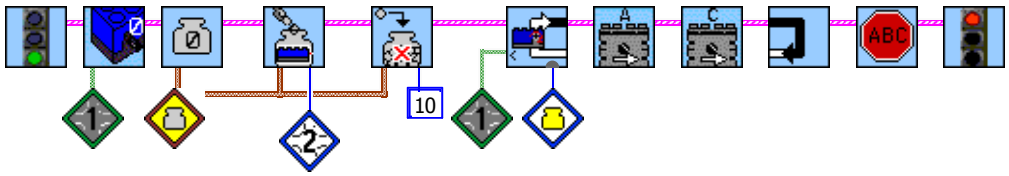
Containers in ROBOLAB are similar to variables that let you store and manipulate values. The container (variable) is represented by color, such as the Red Container icon, while the stored value of the container is represented by the Value of Red Container icon. You can fill a container (assign a number to the variable) by using the Fill Container icon or by assigning a sensor value using icons such as the Light Container.

Once a value is assigned to a container, it can be manipulated mathematically using icons such as the Add to Container icon. The value can also be retrieved and used later on in the program.

Example…

In this example, the Yellow Container is filled with the value of light sensor in Port 2. The value in Yellow Container is then multiplied by ten. This new value is used in a While loop so that the car runs until value of the rotation sensor (in Port 1) exceeds the value in the Yellow Container. Result: The brighter the room, the farther the car will run.

Note: Rotations sensor needs to be zeroed at the beginning of the program.



Containers are useful for converting a sensor's input (i.e., light) into other tangible forms (i.e., sound, motor movement). For example, using NXT distance sensor, you can create a device that will emit a differently pitched sound depending on the reading from the distance sensor. Such device can be a virtual trumbone, or aid a visually impaired person hear when he/she is getting close to a wall.

RCX COMMUNICATIONS

Some of the icons…

 Start Direct RCX Communication

 End Direct RCX Communication

 Send Mail

 Wait for Mail

Description…

Direct RCX Communication allows two RCX's to communicate with one another. This is useful for synchronizing programs on two or more RCX's.

RCXs communicate using "mail" which sends a number to another RCX. The Start Direct RCX Communication icon must be used before any mail is sent or received. Mail is sent from an RCX using Send Mail icon. The receiving RCX is programmed to do certain actions depending on the value sent by the Mail.

Example…

In this example, two RCXs are used (RCX1 and RCX2). RCX1 will use the following program. The program moves motors A and B forward for 2 seconds, then sends a number "1" to another RCX using Send Mail before stopping the motors.

RCX2 will use the program below. It first waits to receive a number "1" via mail, then runs motors A and B for 2 seconds before stopping.

Running both programs simultaneously, RCX1 will move for 2 seconds, immediately followed by RCX2 moving for 2 seconds.

Mail is useful for large projects where multiple RCX's are used. You can also use RCX Direct Communication commands to run a certain program on a remote RCX.

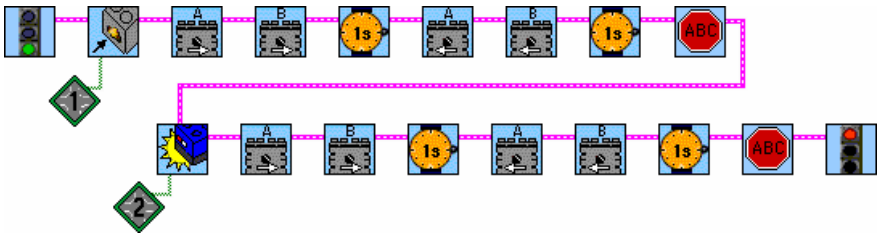
RCX Communication uses the infrared port of the RCX. The infrared ports on the RCXs must be visible from one another when sending/receiving mail.

SubVI's

You can create a "SubVI" in ROBOLAB to make your own functions. SubVI's help break down long programs into smaller, more concise format. Furthermore, once created, SubVI's can be saved and used over and over in another program.

How to make a SubVI…

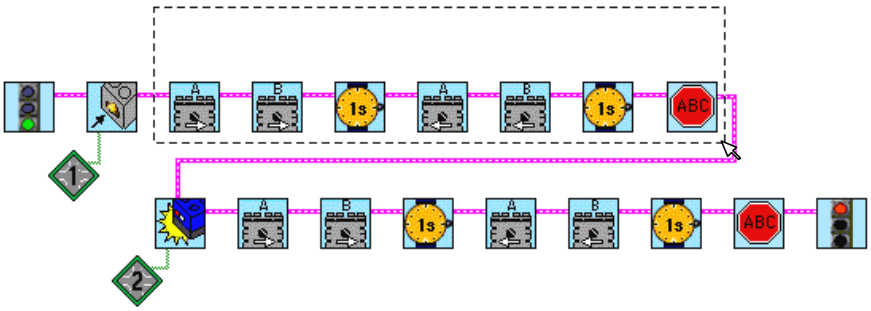
Let's say that you had the following program:



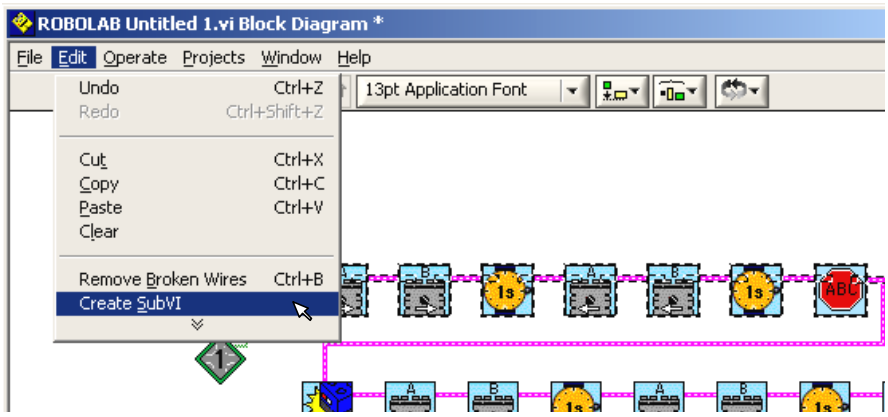
This program 1) waits for the touch sensor to be pressed; 2) moves motors A and B forward for 1 second, reverse for 1 second, and stops; 3) waits for the light sensor to see brighter; then 4) moves motors A and B forward for 1 second, reverse for 1 second, and stops.

Notice that steps 2 and 4 of the program are identical. These steps can be made into a SubVI to make this program considerably shorter.

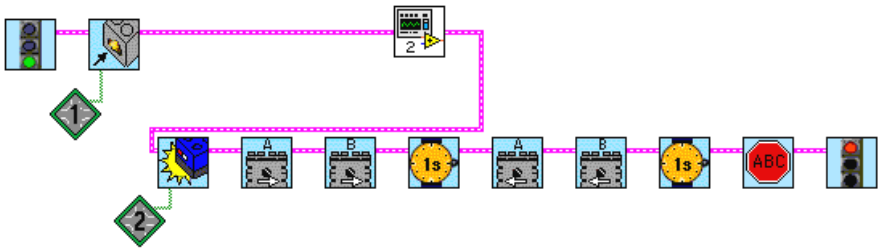
1. First, use the selection tool (the arrow) to draw a box around all icons that are to be included in the SubVI to select them.



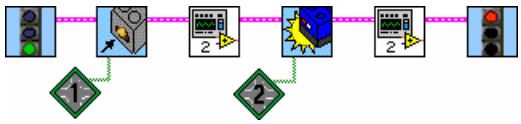
2. Next, while the seven icons are highlighted, go to Edit → **Create SubVI**.



The highlighted icons (all of step 2) are replaced by one icon. This one icon represents all seven icons in step 2.



3. The newly created icon can be copied and pasted to replace icons for step 4. After rearranging, the program looks like:



   The new program is much more concise and simple to read than the original program.

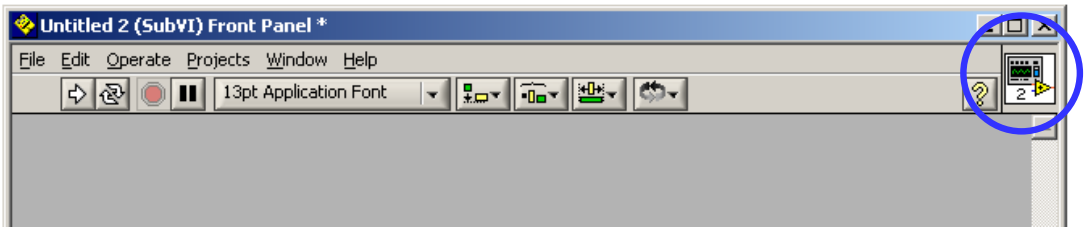
4. To save this SubVI, double-click on the SubVI icon to open the Front Panel. In the new window, go to File → Save As… to save the SubVI.

   You can go to Window → Show Block Diagram to see the code that corresponds to this SubVI (the code that was highlighted in Step 1).

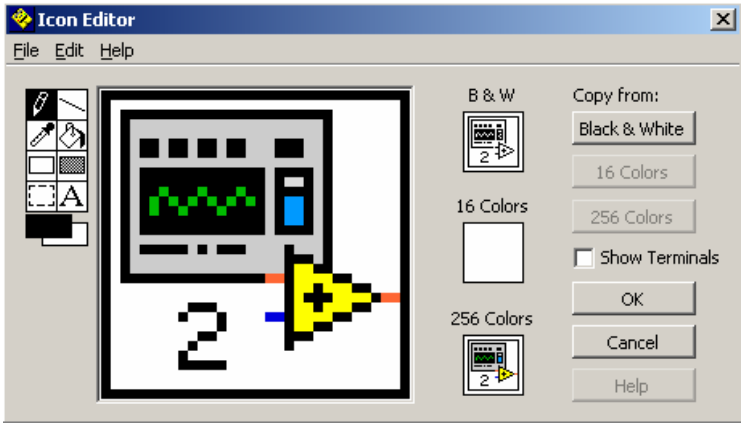
5. To use this SubVI elsewhere, go to the Functions Palette → Select a VI (bottom right corner of Functions Palette), and locate your saved SubVI in the dialog box that opens up.

If you wish, the SubVI icon can be edited so that it is easier for you to use in a program. This is especially important when you have more than one user-made SubVI.

After double-clicking on the icon to open the SubVI, a window containing the SubVI's front panel opens up. The SubVI icon appears on the top right of this window.



Double-clicking on this icon opens up the Icon Editor window which allows you to change the icon to your liking. Use drawing tools on the left side of the window to edit the icon.



SUBROUTINES

Some of the icons...


Create Subroutine


Run Subroutine


Delete Subroutine

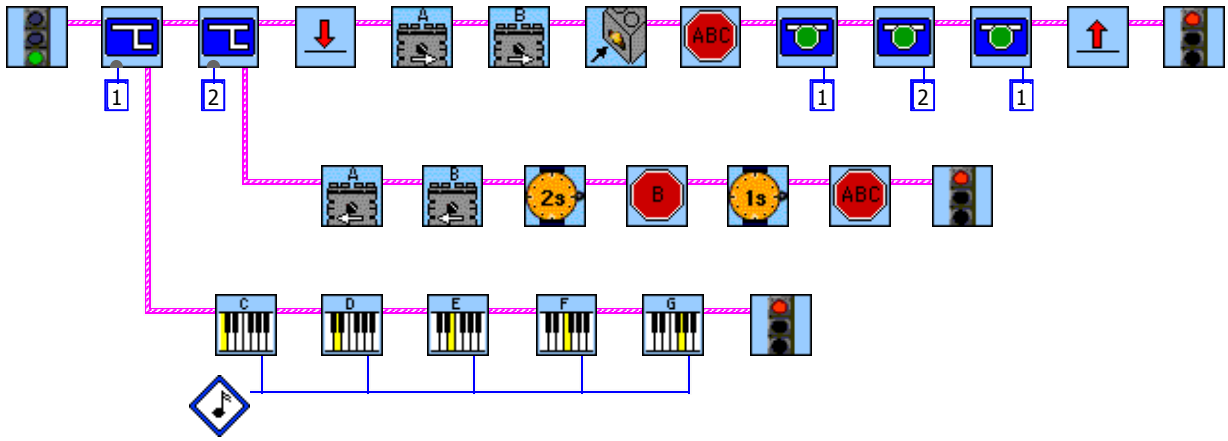
Description...

Like a SubVI, Subroutines are used to make long programs more concise and organized. Unlike a SubVI, however, subroutines are written within a program and can only be used in that program. Create Subroutine icon is used to make a subroutine. You can have up to eight subroutines in a program, and they are differentiated using a number (0~7). Run Subroutine icon is used to run the subroutine at any later point in the program.

Example...

In this program, two subroutines are created. Subroutine 1 plays five notes. Subroutine 2 runs motors A and B in reverse for two seconds, stops motor B (while running motor A) for 1 second, and stops all motors. Note that these subroutines do not actually execute at the time they are made.

The main program (top line) turns on motors A and B. It waits for a touch sensor (i.e., a bumper) to be pressed then stops all motors. The Run Subroutine icon is used to run subroutine 1, subroutine 2, then subroutine 1 again. The entire program repeats using Jump/Land icons.



When this program is used on a car with two motors and a bumper, the car will move forward until the bumper hits a wall. The car will then play five notes (Subroutine 1), back up for two seconds and turn for one seconds (Subroutine 2), and play five notes again (Subroutine 1) . It then goes forward again and repeats forever.

Subroutines are useful for organizing long programs. You can also use subroutines when you have sections of the program that appear more than once in the program. Rather than repeating the code, you can write it once as a subroutine and insert it multiple times within a program.